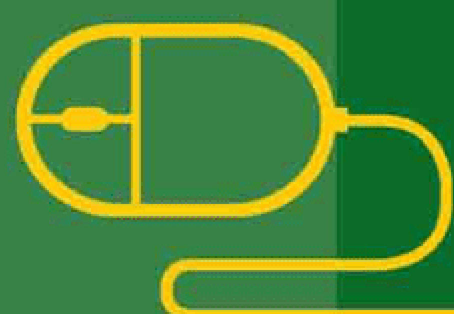
资深软件开发专家多年经验结晶，深入浅出阐释Office VBA开发涉及的工具、方法和实践

由浅入深剖析Office VBA开发过程中遇到的各个层面的问题，涉及文件和路径操作、自定义Office界面、正则表达式、字典、数据库与SQL、跨Office组件编程、工程引用与外部对象、网页自动化等核心内容

Office VBA开发经典



中级进阶卷

刘永富 刘行◎著



清华大学出版社

Office VBA开发经典

中级进阶卷

刘永富 刘行◎著

清華大学出版社
北京

内 容 简 介

资深软件开发专家根据自己十余年 VBA 开发经验，通过本书深入阐释 Office VBA 开发。相比于基础入门卷，本书的内容体系更加完善，知识点更高阶，以 VBA 中添加和使用外部引用为主线，详细讲述使用 VBA 操作和读写 Office 文档之外的内容，案例丰富，让读者身临其境，体会 VBA 编程的策略和魅力。

本书内容丰富、实用性强，实例典型且有代表性，可以帮助读者轻松熟悉 VBA 编程，系统学习 VBA 编程的每个层面。全书分为 14 章，内容包括文件和路径操作、文件系统自动化、压缩文件处理、XML 操作、自定义功能区、正则表达式使用方式、字典使用方法、数据库操作、Office VBA 混合编程、工程引用与外部对象、Acrobat 对象操作、邮件处理、网页自动化等。书中所有章节涉及的程序代码都给出了详细注释。

本书可作为职场办公人员、高校理工科师生、Office 专业开发人员自学用书，也可作为 Office 编程培训讲师的教学参考书。

图书在版编目 (CIP) 数据

Office VBA 开发经典——中级进阶卷 / 刘永富，刘行著. —北京：清华大学出版社，2019
ISBN 978-7-302-52637-7

Ⅰ. ① O… Ⅱ. ①刘… ②刘… Ⅲ. ① BASIC 语言—程序设计 Ⅳ. ① TP312.8

中国版本图书馆 CIP 数据核字（2019）第 046879 号

责任编辑：秦　健
封面设计：李召霞
责任校对：徐俊伟
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm × 260mm　　**印　　张：**26.5　　**字　　数：**612 千字
版　　次：2019 年 7 月第 1 版　　**印　　次：**2019 年 7 月第 1 次印刷
印　　数：1 ~ 2000
定　　价：99.00 元

产品编号：080251-01

Foreword 前　言

Office VBA 编程在全世界范围已经非常普及和流行，办公人员轻轻地按下【 Alt+F11 】快捷键，通过录制宏就能开启 VBA 编程之旅。

VBA 的最简单应用莫过于循环处理工作簿、工作表和单元格数据。然而，Excel 中的数据只是日常办公的一小部分内容，很多数据存储在各种各样的文件中，例如记事本文件、CSV 文件、PDF 文件、数据库等，我们不得不每天双击打开一个文件，手工编辑修改，然后关闭保存。随着大数据时代、信息化时代的迅速发展，自动化、智能化办公成为大势所趋。为此，我们要对手动办公说："No！"

作者根据自身工作经历，深切体会到文件系统自动化的必要性和重要性，用 VBA 操作 Excel 对象还不能充分发挥 VBA 的魅力。因此，本书以文件（Files）为操作对象，以外部引用（Reference）为操作手段，深刻阐述：

- 文本文件的读写。
- 文件、路径的增删。
- 注册表的读写。
- 压缩和解压缩。
- XML 与 Office 界面。
- 正则表达式处理字符串。
- 字典的特色和作用。
- ADO 访问数据库、SQL 查询语句。
- 不同 Office 组件的互相访问和通信。
- 工程引用与外部对象。
- Acrobat 软件和 PDF 文档自动化。
- 自动发送邮件。
- 网页自动化和网页数据获取。

这是进一步提高 VBA 编程技术的必经之路。

本书的组织结构

全书共分为 14 章。

第 1 章介绍文本文件的多种读写方式、路径和文件的自动化处理。

第 2 章介绍自动启动其他应用程序的方法、注册表的读写、环境变量的读写方法。

第 3 章分别介绍用 WinRAR 和 Shell32 自动处理压缩文件，以压缩文件的视角认识 Office 文档。

第 4 章介绍 XML 的语法规则，使用 XML DOM 对象读写 XML 文件，XML 文件的验证等。该章是 Office 自定义界面的理论基础。

第 5 章介绍 Office 界面五大功能区的 customUI 技术，详细讲解了用于定制 Office 界面各种控件的技术要点。

第 6 章介绍 VBA 编程中正则表达式的用法，通过丰富的实例演示和体现了正则表达式在 VBA 编程中的强大之处。

第 7 章介绍字典的构成和特点，详细讲解键值对的添加、删除、修改、检索的技巧和注意点。

第 8 章介绍使用 ADO 技术访问 Access、Excel 数据库，讲解最常用的 SQL 语句。

第 9 章介绍跨 Office 组件编程，讲解前期绑定和后期绑定的区别，以及不同 Office 组件之间的互相访问。

第 10 章介绍 VBA 工程中引用的自动添加和移除，外部对象的创建方法。

第 11 章介绍使用 VBA 调用 Acrobat 对象，对 Acrobat 软件以及 PDF 文件进行读写。

第 12 章介绍邮箱的 SMTP 配置、使用 CDO 对象自动创建和发送邮件。

第 13 章介绍 HTML DOM 读写网页，WebBrowser 和 Internet Explorer 实现网页自动化，以及 XMLHTTP 和 WinHttp 实现 HTTP 请求。

第 14 章介绍 VBA 编程中遇到的其他常见话题：随机数、颜色和进制转换、日期时间方面的计算。

本书的特点

- 编排合理、内容丰富。
- 针对性的实例比较多，知识点讲解透彻。
- 实用性强，讲解的知识范围面向广大计算机办公人员所需。

本书的读者对象

- 职场中使用计算机的办公人员。
- 各类学校的教师、学生、科研人员。
- 从事 VBA 开发、VSTO 开发的相关人员。

本书使用环境

本书的写作环境为 Windows 7（32 位）+ Microsoft Office 2013。不过本书内容在 Office 2010 及其以上版本均兼容。

配套资源

本书配套资源包括：



- ❑ 书中涉及的所有实例文档。
- ❑ 开发资源（编程过程中用到的工具、软件）。

扫描右侧二维码访问上述资源。

读者服务



为方便广大读者学习和探讨，读者可以通过扫描右侧二维码获得更多信息。

建议学习方法

本书各章内容具有相当的难度和深度，其中第 1、2、5 ～ 9 章为必学内容，其余 7 章为选修内容。读者可根据自身所需和兴趣选择学习。

对于书中讲述的每个知识点，读者应清楚其目的和意义，实现的思路和方法，运行结果的分析等环节。

在实际学习过程中，读者可以从书中找到对应的实例文档，直接调试、运行范例中的宏，更便捷地体会程序设计的魅力。

致读者

VBA 是微软 Office 软件配套的一门编程语言，是处理文档、电子表格的首选语言。实际上，VBA 不光能处理、解决 Office 方面的问题，通过添加外部引用，还可以操作和读写 Office 以外的很多内容。外部对象库的引入使得 VBA 如虎添翼，使用恰当的外部引用解决实际问题，让编程变得更加简单、专业。

作者根据自身多年的学习和研究经验，把平时用到最多的外部引用和对象以具体实例的形式编成此书。

本书从立意到写作、交稿历时一年之久，融入作者大量精力和心血。衷心希望广大读者能够从本书汲取营养、查漏补缺，达到学以致用的效果。

致谢

在本书的编写过程中，除了刘永富、刘行外，参与编写的人员还有戴海东、曹文丽、李白等。在编写过程中难免会有疏漏之处，欢迎读者通过清华大学出版社网站 www.tup.com.cn 与我们联系，帮助我们改正提高。

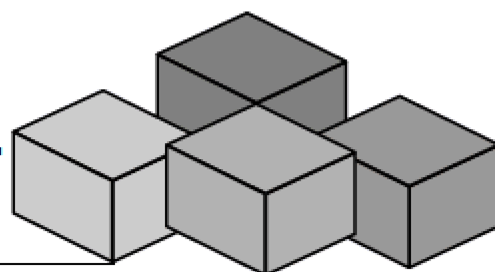
在本书的出版过程中，得到了清华大学出版社策划编辑秦健的大力支持和配合，在此表示衷心感谢。另外，本书所有的编审、发行人员为本书的出版和发行付出了辛勤劳动，在此一并致谢。

刘永富

Contents 目　录

第 1 章

文件和路径操作



我们每天使用计算机处理最多的就是文件和路径这两类对象，从 Visual Basic 的第一版至今，VB/VBA 中有关文件的处理都是通过使用 Open、Write 以及其他一些相关的语句和函数来实现的。

随着软件技术的不断发展，加上面向对象编程概念的日臻成熟，这些文件操作语句已经不能适应软件不断增加的复杂程度的需要。因此，从 VB6.0 开始，微软提出了一个全新的文件系统对象 FileSystemObject（简称 FSO）。

本章主要介绍通过如下三种方式来处理文件和路径。

- 使用传统方式。
- 使用文件系统对象 FileSystemObject（FSO）对象。
- 使用 Shell 语句调用 DOS 命令。

本章用到的外部引用和重要对象如下。

- Microsoft Scripting Runtime
- Scripting.FileSystemObject
- Scripting.TextStream
- Microsoft ActiveX Data Objects 2.8 Library
- ADODB.Stream
- Microsoft XML, v6.0
- MSXML2.XMLHTTP60

1.1 使用传统方式

使用传统方式可以访问文件和路径，对文本文件和二进制文件进行读写。最常用的函数和命令如下。

- Dir：用于列举路径下的文件和子文件夹名称。

- ❑ GetAttr 和 SetAttr：获取和设置属性。
- ❑ FileCopy、Name、MkDir 等：对文件和路径复制、移动等。
- ❑ Open...Write...Close：对文本文件、二进制文件进行打开、读写、关闭。

1.1.1 获取文件或路径的属性

右击文件、文件夹，在弹出菜单中选择属性命令，打开属性窗口后，可以设置只读属性和隐藏属性等。

GetAttr 函数用来获取和判断文件或路径的属性，该函数的参数是一个路径字符串，返回值是由多个 2 的整数幂的组合相加的总和，如表 1-1 所示。

表 1-1 文件、路径的属性常量

常　数	值	描　述
vbNormal	0	常规
vbReadOnly	1	只读
vbHidden	2	隐藏
vbSystem	4	系统文件
vbDirectory	16	目录或文件夹
vbArchive	32	上次备份以后，文件已经改变
vbalias	64	指定的文件名是别名

这里假定磁盘下的 TE.txt 文本文件已设置为“只读”并且“隐藏”，如图 1-1 所示。



图 1-1 查看文件属性

此时，GetAttr("C:\temp\abcd\TE.txt") 会返回一个整数 35。其实，35=32（vbArchive）+2（vbHidden）+1（vbReadOnly）。

因此，把 GetAttr 函数的计算结果拆分为多个枚举常量值之和，就可以得知该文件的属性。

下面的过程用来把任何一个正整数拆分为多个 2 的乘方。

```
Sub SplitAttributes()
    n = 13
    Do Until n = 0
        i = 2 ^ Int(Log(n) / Log(2))
        n = n - i
        Debug.Print i
    Loop
End Sub
```

运行上述过程，可以看到 13 被拆分为 8+4+1。根据这个思路，可以设计一个用来判断文件是否被设置为只读的自定义函数。

```
Function IsReadOnly(Path As String) As Boolean
    n = GetAttr(Path)
    Do Until n = 0
        i = 2 ^ Int(Log(n) / Log(2))
        n = n - i
        If i = VBA.VbFileAttribute.vbReadOnly Then
            IsReadOnly = True
            Exit Function
        End If
    Loop
End Function
```

这个函数的原理就是把 GetAttr 的结果拆分为多个数字，拆分的过程中，看看是否有一个拆分恰好等于枚举常量 vbReadOnly，如果有就提前退出函数，返回 True。

运行 Debug.Print IsReadOnly("C:\temp\abcd\TE.txt")，在立即窗口返回结果 True，表明这是一个只读文件。

同理，把上述函数中的 ReadOnly 这个单词替换为 Hidden，就形成了可以判断文件或路径是否设置了隐藏属性。

```
Function IsHidden(Path As String) As Boolean
    n = GetAttr(Path)
    Do Until n = 0
        i = 2 ^ Int(Log(n) / Log(2))
        n = n - i
        If i = VBA.VbFileAttribute.vbHidden Then
            IsHidden = True
            Exit Function
        End If
    Loop
End Function
```

这里假定 C: 盘下的 Build 文件夹被设置了隐藏属性，那么 Debug.Print IsHidden("C:\Build\") 返回结果 True。

下面的函数可以判断一个路径是否为文件夹。

```
Function IsDirectory(Path As String) As Boolean
    n = GetAttr(Path)
    Do Until n = 0
        i = 2 ^ Int(Log(n) / Log(2))
        n = n - i
        If i = VBA.VbFileAttribute.vbDirectory Then
            IsDirectory = True
            Exit Function
        End If
    Loop
End Function
```

Debug.Print IsDirectory("C:\Build\") 返 回 True。Debug.Print IsDirectory("C:\Build\Hello.csv") 返回 False。

1.1.2 设置文件或路径的属性

与 GetAttr 相对应的函数是 SetAttr，该函数可以设置文件、路径的属性。

```
SetAttr "C:\Build\", vbHidden + vbReadOnly
```

这条代码把 Build 文件夹的属性设置为只读，并且隐藏。

```
SetAttr "C:\Build\", vbNormal
```

这句代码去掉只读和隐藏属性，恢复为正常属性。

1.1.3 判断文件或路径是否存在

使用 Dir 函数可以列举出当前路径下所有文件和子文件夹的名称，从而间接地判断一个文件或文件夹是否存在。

Dir 函数的语法为：

```
Dir(PathName,Attributes)
```

PathName 是一个用来描述文件、路径的字符串，可以使用 *、？ 通配符。Attributes 可以使用如表 1-2 所示的值。

表 1-2　Dir 函数用的筛选常量

常　　数	值	描　　述
vbNormal	0	（默认）指定没有属性的文件
vbReadOnly	1	指定无属性的只读文件
vbHidden	2	指定无属性的隐藏文件
VbSystem	4	指定无属性的系统文件
vbVolume	8	指定卷标文件，如果指定了其他属性，则忽略 vbVolume
vbDirectory	16	指定无属性文件及其路径和文件夹

如果不规定 Attributes 属性，则默认为 vbNormal。

```
Sub 判断文件或路径是否存在 ()
    Path = "C:\temp\Test.txt"
    If Dir(Path) = "" Then
        Debug.Print Path & " 不存在 "
    Else
        Debug.Print Path & " 确实在计算机中。"
    End If
End Sub
```

代码分析：如果计算机中不存在 Test.txt 这个文件，那么 Dir 函数会返回空字符串；如果文件存在，则返回第一个符合模式的文件名称（不包含所在路径），据此可以判断磁盘或文件夹中是否有某个文件。此外，还可以使用 Dir 函数判断是否有某磁盘分区，或者是否有某个文件夹。

如果上述过程中的 Path 赋值为 Path="M:" 或者 Path="M:\"，则可以用来判断是否存在 M: 盘。

如果要判断是否存在某文件夹（路径），结尾必须加反斜杠。例如 Dir("C:\build") 用来判断 C: 盘下是否有 build 这个文件，而 Dir("C:\build\") 用来判断 C 盘下是否有 build 文件夹。

1.1.4　遍历文件和子文件夹

利用 Dir 函数和不带参数的 Dir，可以遍历一个路径下的所有文件和子文件夹的名称。现在假定 C:\CTEX 文件夹中的内容如图 1-2 所示。

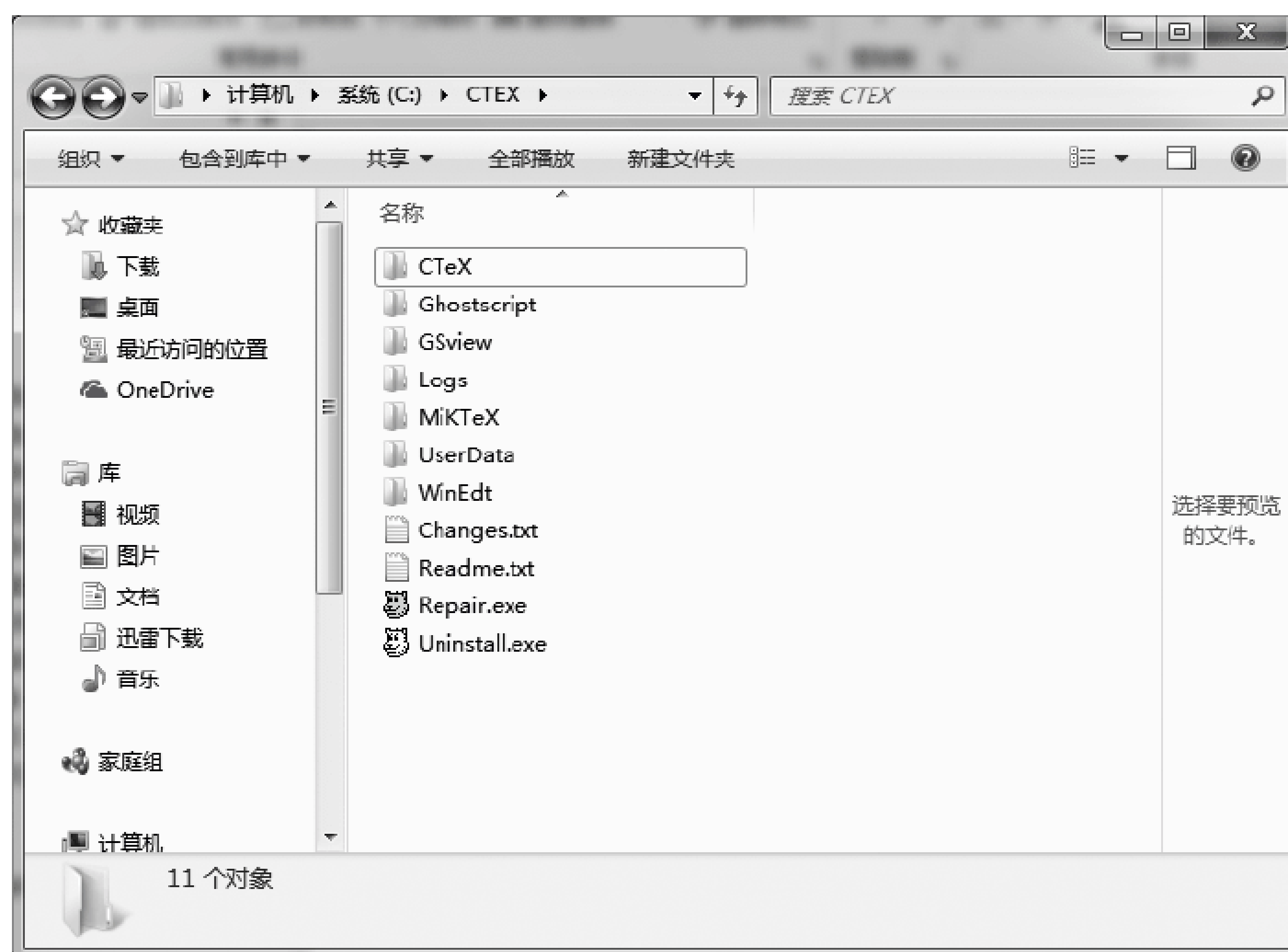


图 1-2　文件夹中的内容

可以看到有 7 个子文件夹，4 个文件。运行如下的过程，打印出所有的子文件夹名称和文件名称。

```
Sub 遍历文件和子文件夹 ()
    Dim parent As String, Path As String
    parent = "C:\CTEX\"
```

```
        Path = Dir(parent, vbDirectory)
        Debug.Print Path
        Do Until Path = ""
            Path = Dir
            Debug.Print Path
        Loop
    End Sub
```

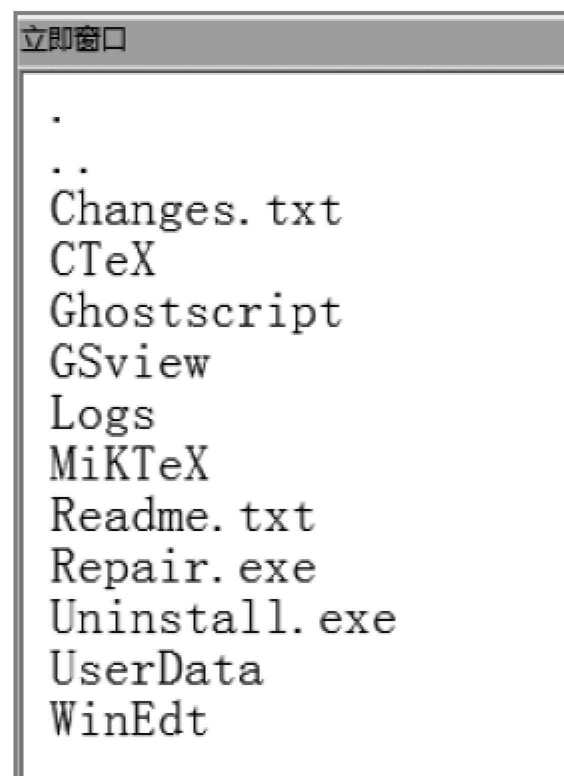


图 1-3 遍历子文件夹和文件

上述程序的打印结果如图 1-3 所示。

可以看出，第一行打印出一个小数点，第二行打印出两个小数点，从第三行起才是正式的内容。

如果把代码中的 Path = Dir(parent, vbDirectory) 修改为 Path = Dir(parent)，则只遍历文件，不遍历子文件夹。

那么如何只遍历子文件夹呢？这就需要在循环体中加入 If 语句来选择性地遍历。

```
Sub 遍历子文件夹 ()
    Dim parent As String, Path As String
    Dim Col As New Collection, v As Variant
    parent = "C:\CTEX\"
    Path = Dir(parent, vbDirectory)
    Col.Add Path
    Do Until Path = ""
        Path = Dir
        Col.Add Path
    Loop
    For Each v In Col
        If GetAttr(parent & v) And vbDirectory Then
            If v = "." Or v = ".." Then

            Else
                Debug.Print v
            End If
        End If
    Next v
End Sub
```

上述过程中，用集合 Col 来装载所有的文件和子文件夹的名称，最后，遍历 Col 的时候，首先过滤出所有的子文件夹，然后排除小数点，最后输出纯粹的子文件夹，共 7 个，如图 1-4 所示。

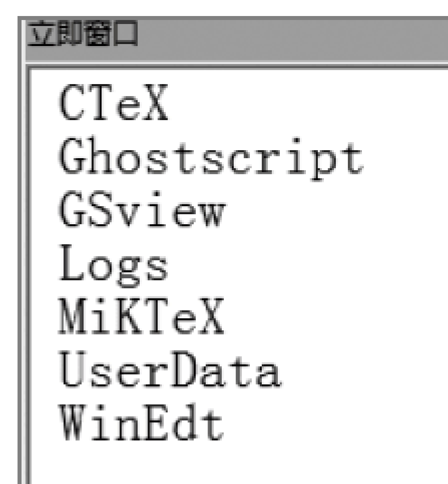


图 1-4 只列举子文件夹名称

如果要遍历 C:\Ctex 下面的所有扩展名为 .txt 的文本文件，代码可以修改为如下形式。

```
Sub 遍历文本文件 ()
    Dim parent As String, Path As String
    parent = "C:\CTEX\"
    Path = Dir(parent & "*.txt")
    Debug.Print Path
    Do Until Path = ""
        Path = Dir
```

```
            Debug.Print Path
        Loop
End Sub
```

注意，Dir 函数中用到了通配符，*.txt 可以匹配所有的文本文件，如图 1-5 所示。

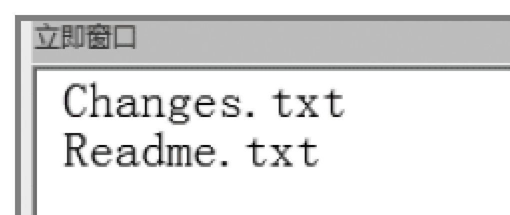


图 1-5　只遍历文本文件

1.1.5　文件的复制、移动和删除

文件的复制、移动和删除操作，分别用 FileCopy、Name As 和 Kill 语句。

FileCopy 的语法为：

```
FileCopy Source, Destination
```

Source 表示原文件，Destination 表示复制的目标。

例如 FileCopy Source:="C:\temp\a.xlsx", Destination:="D:\dist\goal.xlsx"，表示把文件 C:\temp\a.xls 复制到 D:\dist 文件夹下，并且重命名为 goal.xlsx。

文件的移动操作就是文件的剪切，也可以理解为文件的重命名。与复制文件的区别是，原文件不在原位置了。

Name "C:\temp\a.xlsx" As "D:\dist\b.xlsx"，就相当于把 a 文件从原位置剪切到 D:\dist 文件夹中，并且设置名称为 b.xlsx。

注意　针对文件的移动操作，如果 D:\dist\ 下面原先就有一个 b.xlsx 文件，那么运行上述的 Name 语句会导致出错。也就是说，必须保证目标文件夹中还没有这个文件，才能进行移动操作。

Kill 语句用于删除文件，如果文件处于打开、占用状态，运行该语句会出错。另外，用 Kill 语句删除掉的文件，不能通过回收站还原，要谨慎操作。

图 1-6 所示的代码连续两次删除同一个文件，第一句不会出错，但是运行到第二句时弹出“文件未找到”的错误。

```
Sub 删除文件()
    Kill "D:\dist\b.xlsx"
    Kill "D:\dist\b.xlsx"
End Sub
```

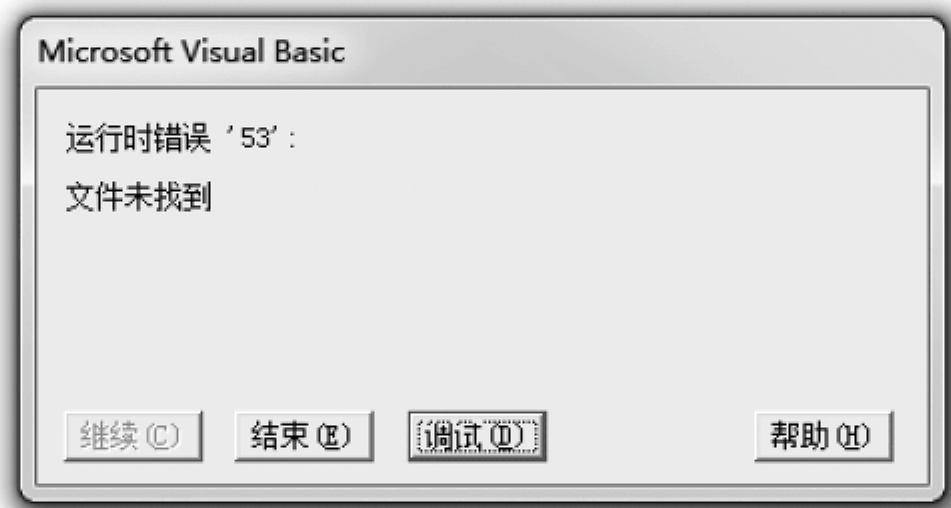


图 1-6　重复删除同一文件的错误

1.1.6 文件夹的创建和删除

文件夹的创建和删除分别用 MkDir 和 RmDir 语句，Mk 是 Make 的缩写，Rm 是 Remove 的缩写。

MkDir 语句的语法很简单。

MkDir Path:="C:\temp\2017"，会在 temp 文件夹下创建一个名为 2017 的文件夹。

RmDir 语句用来删除一个空文件夹。

RmDir Path:="C:\temp\picture"，表示删除 picture 文件夹，如果该文件夹不是空的，包含其他的文件和子文件夹，那么 RmDir 会提示错误，如图 1-7 所示。

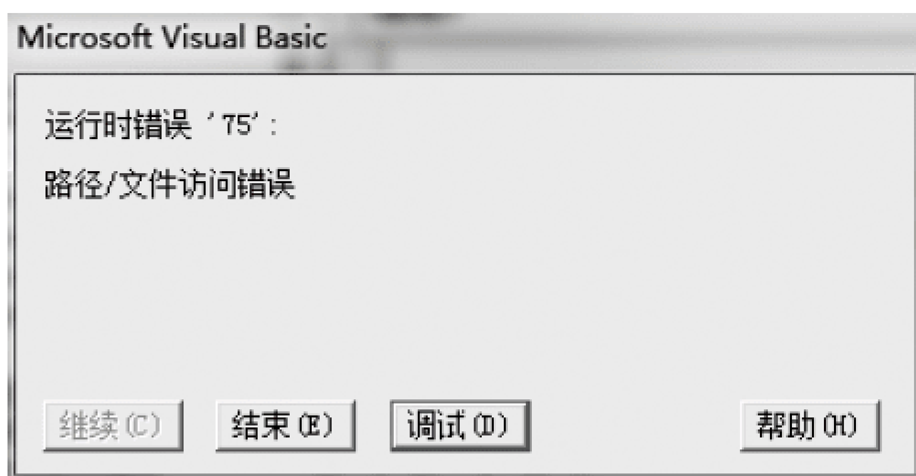


图 1-7 文件夹中有内容则不能删除

也就是说，要删除一个文件夹，必须先把里面的内容清空后，才能使用 RmDir 语句删除。

文件夹的重命名也使用 Name…As 语句。例如 Name "C:\temp\picture" As "C:\temp\pic"，表示把文件夹 picture 重命名为 pic。

1.1.7 文本文件的读写

编程过程中，经常需要把程序运行的结果数据保存到文本文件，也需要从文本文件中读取数据供程序使用，这就涉及文本文件的读写操作了。

本节介绍一下用于文件读写的 Open 语句。

Open 语句的语法如下。

```
Open textFile For mode As fileNum
```

参数 textFile 是一个表示文本文件路径的字符串。

参数 mode 表示 Open 语句的读写模式，使用最多的模式如下。

- ❑ Append：追加模式。
- ❑ Output：擦写模式。
- ❑ Input：读取模式。

如果要把程序运行的结果输出到文本文件中，那么使用 Append 模式会把输出结果追加到文件已有内容之后，而使用 Output 模式，则会先清空文件原先的内容，再写入输出结果。

如果要从文本文件中读取内容，而不破坏文件，可以使用 Input 模式。

要注意的是，在使用 Output 或 Append 模式时，如果计算机中 textFile 文件不存在，则会自动创建一个文本文件；如果使用 Input 模式读取一个文本文件，文本文件不存在会导致出错。

参数 fileNum 是一个文件号，可以是 #1 到 #511 中的任何一个。读写文件操作结束后，

一定要用 Close fileNum 关闭文件。

下面讲述一下导出数据到文本文件中的方法。

```
Sub 导出数据 ()
    Open "C:\temp\abc.txt" For Output As #1
        Print #1, "hello VBA"
        Print #1, "you are welcome"
        Print #1, "2017 年 11 月 25 日 " & vbNewLine & " 刘永富 "
    Close #1
End Sub
```

上述过程把三个字符串写入文本文件中，使用 Print 语句写入时，在末尾自动换行，如图 1-8 所示。

使用 Print 在同一行输出多个字符串时，每个字符串之间用半角分号隔开。

```
Sub 导出数据 ()
    Open "C:\temp\abc.txt" For Output As #1
        Print #1, "hello VBA"; "you are great"
        Print #1, "you are welcome"; "ryueifu"
        Print #1, "2017 年 11 月 25 日 "; " 刘永富 "
    Close #1
End Sub
```

上述程序的运行结果如图 1-9 所示。

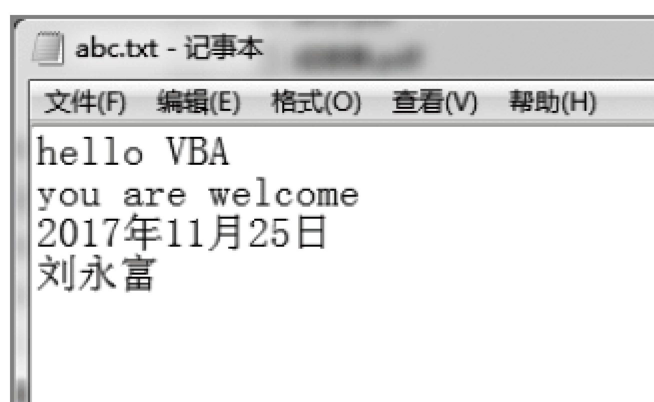


图 1-8　向文件写入内容

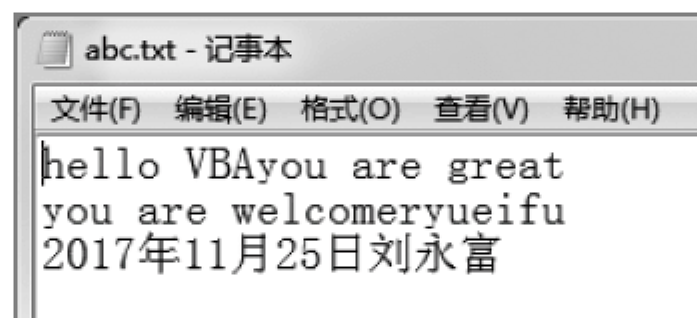


图 1-9　同一行输出多个结果

除了使用 Print 语句输出外，还可以使用 Write 语句输出内容到文本文件。

```
Sub 导出数据 2()
    Open "C:\temp\abc.txt" For Output As #1
        Write #1, "hello VBA"
        Write #1, "you are welcome"
        Write #1, "2017 年 11 月 25 日 "
    Close #1
End Sub
```

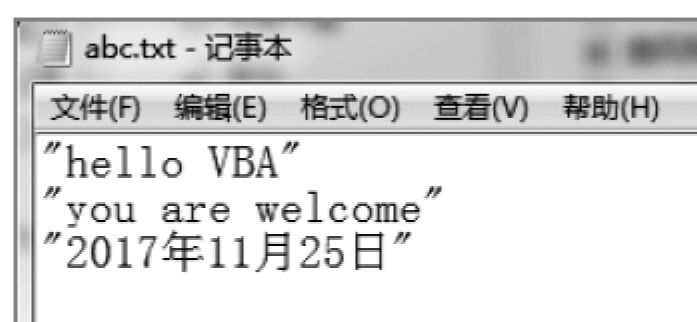


图 1-10　使用 Write 输出内容

程序的运行结果如图 1-10 所示。

可以看出文本文件中的内容都带有双引号，这和 Print 语句有很大不同。

如果把 Open "C:\temp\abc.txt" For Output As #1 这句中的 Output 换成 Append，则每次写入文件时，不删除文件原有内容。请读者自行测试。

接下来讲述如何从已有文本文件中读取内容，供程序调用。

读入文件内容涉及的常用术语有：

- v=Input(c,fileNum)，表示从文件当前位置读取 c 个字符，赋给字符串变量 v。

- ❑ Seek fileNum, c，把当前位置重设为 c，c 的最小值是 1。
- ❑ LOF(fileNum)，返回文件的长度，也就是文件中字符总数。
- ❑ EOF(fileNum)，返回一个布尔值，当读取到文件尾部，返回 True。经常使用 EOF 来判断是否读取完成。

现在假设文本文件 auto.txt 中的内容如图 1-11 所示。

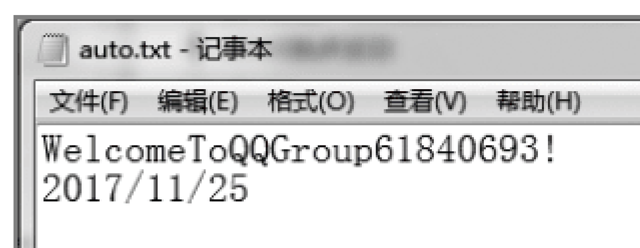


图 1-11　文本文件内容

```
Sub 导入数据 ()
    Dim a, b, c, d, e
    Open "C:\temp\auto.txt" For Input As #1
        MsgBox "文件字符总数：" & LOF(1)
        a = Input(1, #1)
        b = Input(2, #1)
        c = Input(3, #1)
        Seek #1, 1
        d = Input(3, #1)
    Close #1
    Debug.Print a, b, c, d
End Sub
```

代码分析：a = Input(1, #1)，表示从文件的开头处读取 1 个字符，赋给 a，因此变量 a 的取值为字符串 h。

b = Input(2, #1)，表示从上次读取的位置起，读入 2 个字符赋给变量 b，因此 b 的取值为 el。以此类推。

Seek #1, 1 表示把读取位置重设为 1，也就是文件开头，接下来 d = Input(3, #1) 表示从文件开头处读取 3 个字符，因此 d 的取值为 Hel。

上述程序的运行结果如图 1-12 所示。



图 1-12　从文件中读取字符

根据这个特点，可以把文本文件中的所有字符分发到字符串数组中。

```
Sub 文本文件转数组 ()
    Dim s() As String
    Dim i As Long
    Open "C:\temp\auto.txt" For Input As #1
        ReDim s(1 To LOF(1)) As String
        For i = 1 To LOF(1)
            s(i) = Input(1, #1)
        Next i
    Close #1
    Stop
    Debug.Print Join(s, "*")
End Sub
```

代码分析：上述过程，打开文本文件后，根据文件字符总数重新定义数组的上下界，使得数组能恰好容纳文本中的字符，然后使用 For 循环，遍历文本文件中的每个字符，并分发到数组的每个元素。

运行到 Stop 那句，通过本地窗口可以清晰地看到数组 s 的各元素取值情况，如图 1-13 所示。

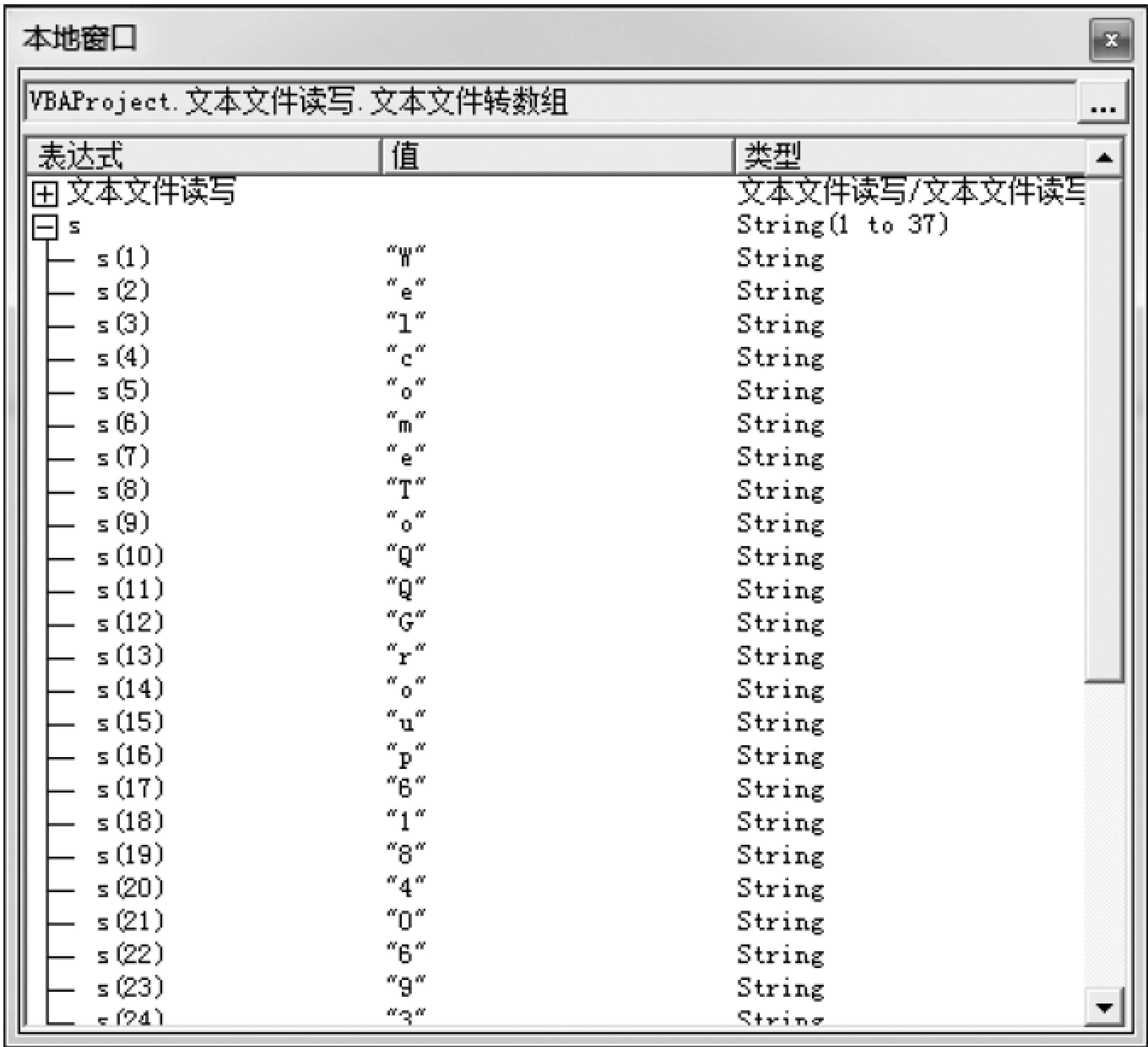


图 1-13　本地窗口查看数组

可以看出，每个元素恰好取得文件中的一个字符。最后通过 Join 把数组用 * 重新连接并输出，如图 1-14 所示。

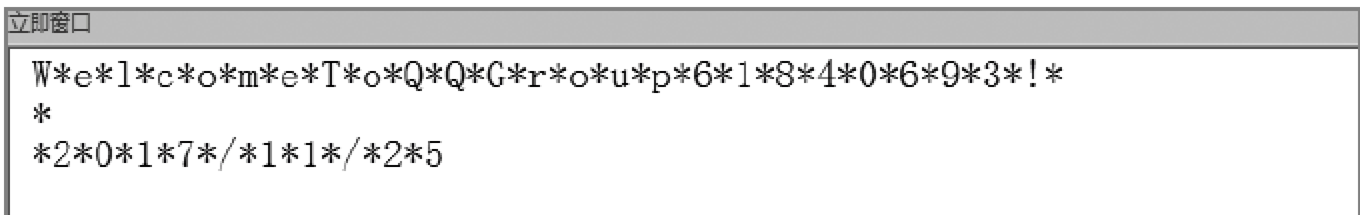


图 1-14　数组连接为字符串

此外，还可以使用 Line Input 语句，每次读取一整行。

假设文件 b.txt 中有 4 行古诗，下面用 Line Input 读取内容。

```
Sub 整行读取()
    Dim s As String
    Open "C:\temp\b.txt" For Input As #1
        Do Until EOF(1)
            Line Input #1, s
            Debug.Print s
        Loop
    Close #1
End Sub
```

代码分析：本例直接把读取到的每行打印到立即窗口，因此可以使用 Do 循环，利用 EOF 函数来判断是否读到文件尾部，如果到了尾部，就结束循环。

上述程序的运行结果如图 1-15 所示。

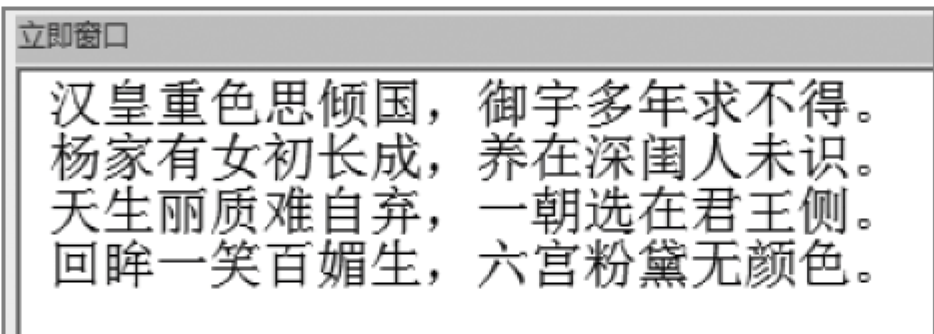


图 1-15　使用 Line Input 读取内容

如果要一次性读取文本文件的所有内容，可以使用下面的自定义函数。

```
Public Function GetAllText(FileName As String) As String
    Open FileName For Binary As #1
        GetAllText = Input(LOF(1), #1)
    Close #1
End Function
```

运行下面的 Test 过程，即可把文件中的所有内容打印到立即窗口。

```
Sub Test()
    Debug.Print GetAllText("C:\temp\new.txt")
End Sub
```

上述代码的源文件为“实例文档 01.xlsm”。

1.2 二进制方式读写文件

计算机中的文件都是以二进制方式存储的，文本文件可以用文本编辑软件查看和编辑，本质上也是以字节为单位存储在磁盘上。

因此，学习用二进制方式读写文件，有助于理解文件和字符串的关系。同时以字节数组这种数据类型为媒介，可以方便地对文本文件进行拆分和合并。

1.2.1 字符串与字节数组的互换

在 VBA 中，String 和 Byte 类型的数组可以通过 StrConv 函数互相转换，当字符串中的每个字符是英文字母或半角字符时，一个字符占据 1 个字节（Byte）；当出现中文汉字或全角字符时，一个字符会拆分为两部分，也就是占据 2 个字节。

下面的代码把字符串“ VBA 学习”转换为字节数组，并在立即窗口打印字节数组的信息，最后再把字节数组转换为字符串。

```
Sub String2Byte()
    Dim s As String
    Dim b() As Byte
    b = VBA.StrConv("VBA 学习 ", vbFromUnicode)
    Debug.Print UBound(b) - LBound(b) + 1
    Debug.Print b(0), b(1)
    s = VBA.StrConv(b, vbUnicode)
    Debug.Print s
End Sub
```

代码分析：“ VBA 学习”共 5 个字符，但由于后 2 个是汉字，所以转换为字节数组后，数组长度为 7，分别从 b(0) 到 b(6)，如图 1-16 所示。

V	B	A	学		习	
86	66	65	209	167	207	176
b(0)	b(1)	b(2)	b(3)	b(4)	b(5)	b(6)

图 1-16 字符串与字节数组的关系

在运行上述代码过程中，通过本地窗口可以看到字节数组 b 的构成，如图 1-17 所示。

本地窗口

VBAProject.二进制方式读写文件.String2Byte

表达式	值	类型
⊞ 二进制方式读写文件		二进制方式读写文件/二
s	""	String
⊟ b		Byte(0 to 6)
b(0)	86	Byte
b(1)	66	Byte
b(2)	65	Byte
b(3)	209	Byte
b(4)	167	Byte
b(5)	207	Byte
b(6)	176	Byte

图 1-17　字节数组的构成

上述过程运行完毕后，在立即窗口打印出字节数组的长度以及每个元素的值，并且把字节数组恢复为字符串，如图 1-18 所示。

```
立即窗口
 7
 86            66
VBA学习
```

图 1-18　运行结果

1.2.2　文本文件的写入

二进制方式打开文件的语句是：

```
Open Path For Binary Access Write As #1
```

使用 Put 语句向文件中写入内容。Put 语句的语法格式为：

```
Put #1,Pos,data
```

其中，#1 是文件号，Pos 用于指定文件的写入位置，如果不指定该参数，则默认在文件结尾处写入。data 是要写入的内容，可以是字符串或变量，也可以是字节数组。

下面的程序演示了二进制方式向文件中先后写入两个字符串。

```
Sub WriteToFile()
    Dim s1 As String, s2 As String
    Dim b1() As Byte, b2() As Byte
    s1 = "VBA 程序 "
    s2 = " 设计 "
    b1 = VBA.StrConv(s1, vbFromUnicode)
    b2 = VBA.StrConv(s2, vbFromUnicode)
     Open "C:\Temp\Example1.txt" For Binary Access
Write As #1
        Put #1, , b1
        Put #1, , b2
    Close #1
End Sub
```

上述代码执行后，记事本中的内容为“VBA 程序设计”，查看该文件的属性，可以看到文件大小为 11 字节。因为“程序设计”4 个汉字占据 8 字节，如图 1-19 所示。

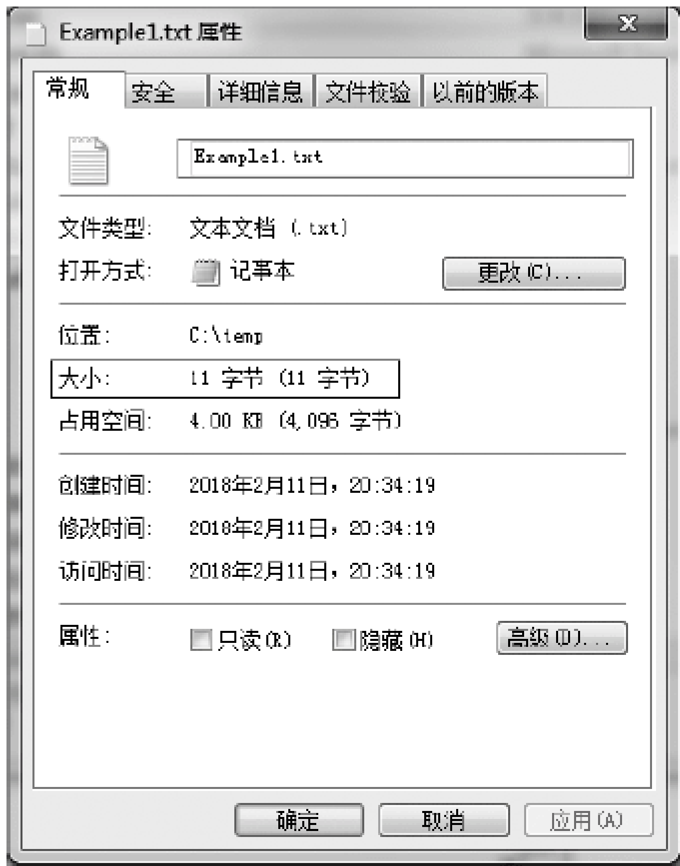


图 1-19　文件的大小

1.2.3　文本文件的读出

二进制方式读取文件的语句是：

```
Open Path For Binary Access Read As #1
```

使用 Get 语句从文件中读入内容。Get 语句的语法格式为：

```
Get #1,Pos,Byte()
```

其中，Pos 参数用来指定读取位置，如果不指定，则从上次读取完毕的位置开始读。

Byte() 是用来存放读出的数据的字节数组。

在读取的过程中，有两点需要注意：一是使用 Get 读取过程中，读取位置也随之向后移动；二是声明的字节数组的大小决定读出内容的多少。

假设一个文本文件中的内容是“ VBA 程序设计”，共 7 个字符 11 个字节。下面的过程分两次读取其中的内容。

```
Sub ReadFile()
    Dim s1 As String, s2 As String
    Dim b1() As Byte, b2() As Byte
    ReDim b1(0 To 4)
    ReDim b2(5 To 10)
    Open "C:\Temp\Example1.txt" For Binary Access Read As #1
        Get #1, , b1
        Get #1, , b2
    Close #1
    s1 = VBA.StrConv(b1, vbUnicode)
    s2 = VBA.StrConv(b2, vbUnicode)
    Debug.Print s1, s2
End Sub
```

代码分析：上述代码中预先指定了字节数组 b1 可以存放 5 个字节，b2 可以存放 6 个字节。因此，当执行 Get #1, , b1 这句时，b1 将获取文件中的前 5 个字节，也就是“ VBA 程”，b2 接着获取其后的剩余部分。

运行上述代码，立即窗口打印出字符串 s1 和 s2，如图 1-20 所示。

以上分批次读取一个文件的原理，是实现拆分文件的理论基础。

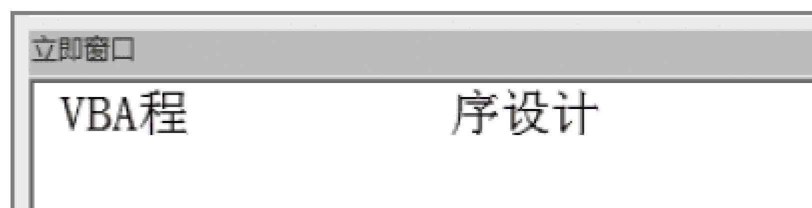


图 1-20　字节数组转换为字符串

在实际编程过程中，经常把文件的所有内容读取到一个字符串变量中，因此改写为如下过程。

```
Sub ReadFile2()
    Dim s1 As String
    Dim b1() As Byte
    Open "C:\Temp\Example1.txt" For Binary Access Read As #1
        ReDim b1(0 To LOF(1) - 1)  '或者 ReDim b1(0 To FileLen("C:\Temp\Example1.
txt") - 1)
        Get #1, , b1
    Close #1
    s1 = VBA.StrConv(b1, vbUnicode)
    Debug.Print s1
End Sub
```

代码分析：要把文件所有内容获取到字节数组中，需要事先知道文件的长度，也就是文

件的字节数，可以用 FileLen 直接获取文件长度，也可以打开文件后用 LOF 函数获取。知道了文件的长度，就可重新定义数组的上下界。

运行上述代码，立即窗口将打印出文件中的所有内容。

1.2.4　文本文件的拆分

如果文本文件中的内容很多，不方便操作时，就需要拆分为若干个小文件，具体拆分为多少个小文件、从什么位置开始拆分，这些基准可以根据实际情况而定。

假设“静夜思 .txt”文本文件里面存放的内容如图 1-21 所示。

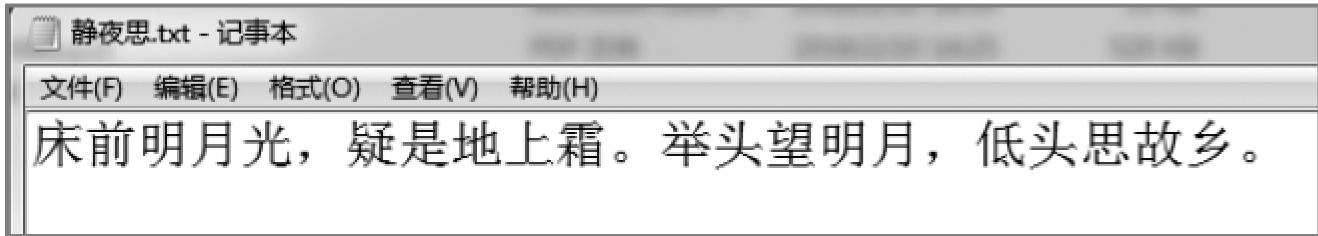


图 1-21　文件内容

可以看到，里面每一句都是 5 个汉字再加 1 个中文标点，也就是说每句占据 12 个字节，整个文件 48 字节。现在要求把这个文件拆分为 4 份，每个小文件保存一句，命名为 Part#.txt。

```
Sub SplitFile()
    Dim b(0 To 11) As Byte
    Dim i As Integer
    Open "C:\Temp\静夜思 .txt" For Binary Access Read As #1
        For i = 1 To 4
            Get #1, , b
            Debug.Print VBA.StrConv(b, vbUnicode)
            Open "C:\Temp\Part" & i & ".txt" For Binary Access Write As #2
                Put #2, , b
            Close #2
        Next i
    Close #1
End Sub
```

代码分析：由于每句占据 12 字节，所以字节数组要规定为 b(0 To 11) As Byte，使得每次恰好读出 12 字节。

然后在循环体中依次读取每一行，并且把每次产生的字节数组保存到不同的子文件中。

上述过程运行后，可以看到文件夹中多了 4 个子文件，如图 1-22 所示。

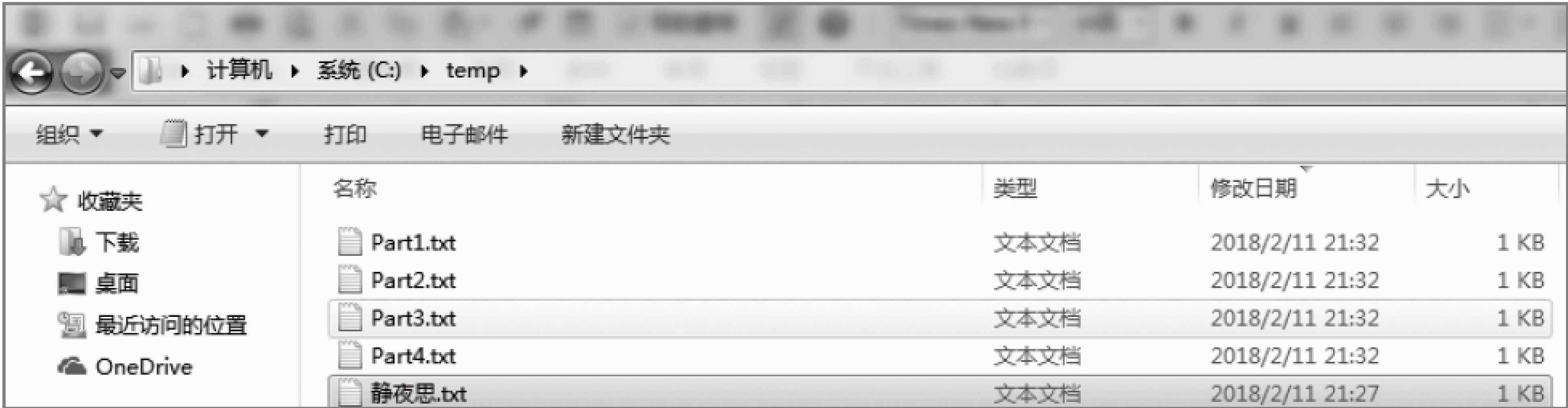


图 1-22　文件的拆分结果

1.2.5 文本文件的合并

平时可能会遇到将很多格式比较相似的记事本文件汇总到同一个文件中的情况。文件合并过程与拆分过程恰好相反，不同的是文件合并过程中，要多次读取每个子文件，然后写入总文件中。

下面的实例在打开总体文件的前提下依次打开每个子文件，读取内容到数组，然后立即写入总文件。

```
Sub MergeFile()
    Dim b(0 To 11) As Byte
    Dim i As Integer
    Open "C:\Temp\Merged.txt" For Binary Access Write As #1
    For i = 1 To 4
        Open "C:\Temp\Part" & i & ".txt" For Binary Access Read As #2
            Get #2, , b
            Put #1, , b
        Close #2
    Next i
    Close #1
End Sub
```

代码分析：总文件的文件号是 #1，子文件的文件号是 #2。

运行上述代码，计算机中产生一个名为 Merged.txt 的新文件，该文件内容是一首完整的“静夜思”。

1.2.6 二进制文件的复制

除了处理文本文件外，还可以使用二进制方式读写计算机中的各种类型的文件，假设读取文件 X 到字节数组 B 中，再把字节数组 B 写入文件 Y 中，这样就实现了文件的复制。

下面的过程把 PythonLogo.png 这个图片读出到数组，然后把数组写入 New.png。

```
Sub CopyFile()
    Dim b() As Byte
    Open "C:\Temp\PythonLogo.png" For Binary Access Read As #1
        ReDim b(0 To LOF(1) - 1)
        Get #1, , b
    Close #1
    Open "C:\Temp\New.png" For Binary Access Write As #2
        Put #2, , b
    Close #2
End Sub
```

运行上述过程后，将在磁盘中多了一个 New.png 图片，这个图片文件与原始图片文件完全相同。

上述代码的源文件为“实例文档 01b.xlsm”。

1.3 使用文件系统对象

FSO（FileSystemObject）不仅可以像使用传统文件操作语句那样实现文件的创建、改变、

移动和删除，而且可以检测是否存在指定的文件夹，如果存在，那么这个文件夹又位于磁盘上的什么位置。更令人高兴的是，FSO 对象模型还可以获取关于文件和文件夹的信息，如名称、创建日期或修改日期等以及系统中使用的驱动器的信息，如驱动器的种类是 CD-ROM 还是可移动磁盘，当前磁盘的剩余空间还有多少。

FSO 对象本身不属于 VBA 对象，要在 VBA 中使用 FSO 操作文件和路径，可以用前期绑定，也可以用后期绑定。

1.3.1　前期绑定

Office VBA 不仅可以使用 VBA 本身的对象、成员，而且可以引入外部对象，例如在 VBA 中使用 FSO、字典、正则表达式，以及后面讲到的操作其他 Office 组件，其实都是在 VBA 工程中引入了外部对象。

所谓的前期绑定，就是在编写程序之前，把外部对象库加入工程的引用（References）中。这样做的好处是，在写代码的时候，这些相关的对象后面输入小数点，可以自动列出成员，而且在声明变量时，也可以直接指定变量的类型。

采用前期绑定方式，可以使用 New 关键字或 GetObject 函数创建一个新的对象。

下面介绍一下采用前期绑定方式，向 VBA 工程引入 FSO 对象的步骤。

单击 VBA 编辑器的菜单【工具 / 引用】，弹出工程的引用对话框。在对话框中勾选 “Microsoft Scripting Runtime”，单击 “确定” 按钮关闭对话框，如图 1-23 所示。

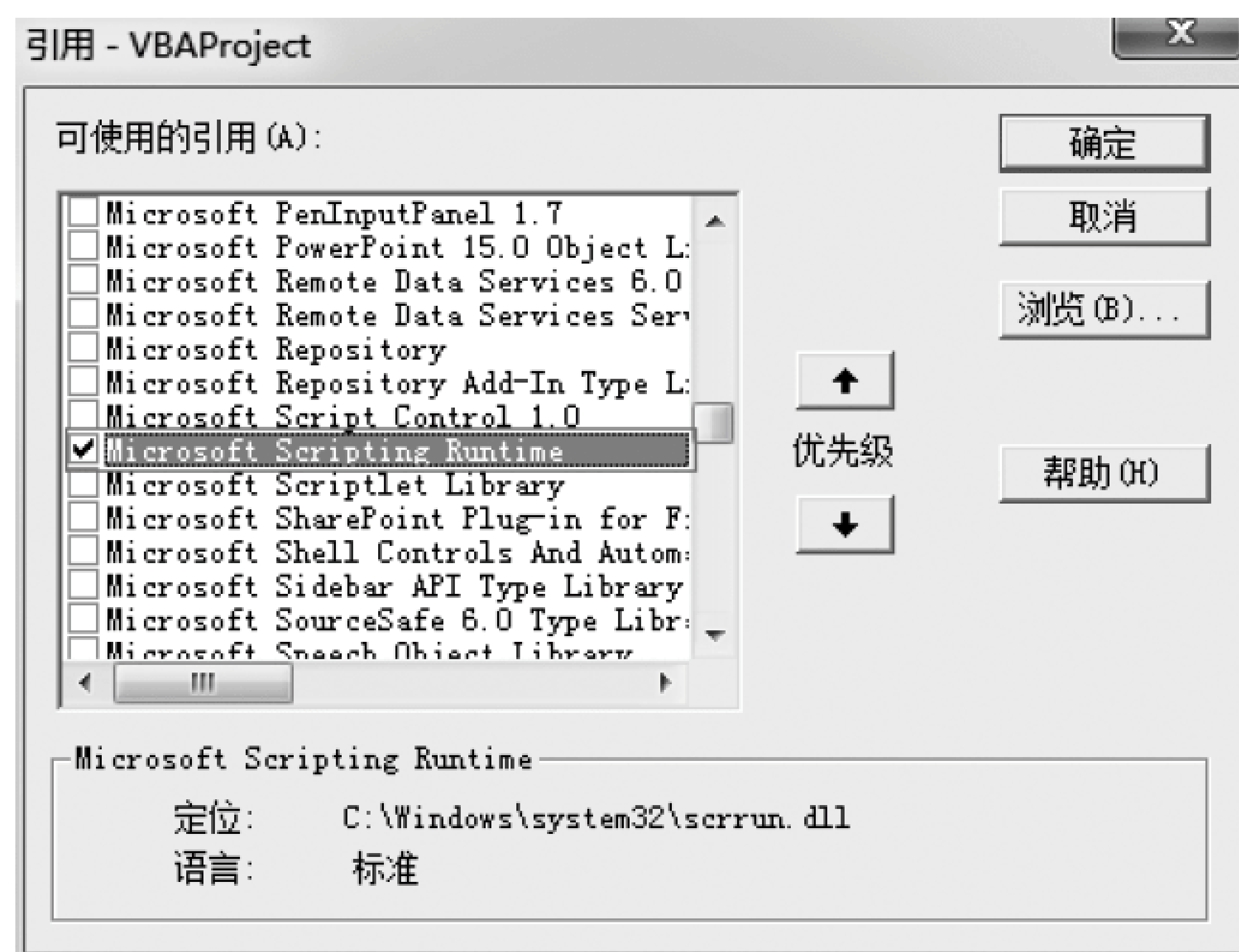


图 1-23　添加外部引用

“Microsoft Scripting Runtime” 这个外部引用位于路径 “C:\Windows\System32\scrrun.dll” 这个动态链接库中，每个 Windows 系统都有这个文件。

VBA 工程一旦引入了这个外部引用，就可以使用 FSO 对象模型，以及后面要讲到的字典（Dictionary）对象。

下面通过一个 VBA 过程来测试一下。

```
Sub Test()
    Dim FSO As Scripting.FileSystemObject
    Set FSO = New Scripting.FileSystemObject
    FSO.CopyFile Source:="C:\dist\34.txt", Destination:="C:\dist\35.txt"
End Sub
```

当输入 Scripting 后面的小数点时，会自动弹出 FSO 相关的成员，这就是前期绑定的特点。运行上述过程，会执行复制文件操作。

1.3.2 后期绑定

后期绑定，就是程序中用到的外部对象，不往工程中添加引用，而是在需要该外部对象的地方，使用 CreateObject 函数来创建对象。针对后期绑定，由于 VBA 的工程没有添加对象库的引用，自然就不会自动列出成员，声明这方面的变量时，只能声明为 Object 或 Variant 类型。

下面新建一个工作簿，打开 VBA 编辑器在标准模块中直接书写一个过程。

```
Sub LateBound()
    Dim FSO As Object
    Set FSO = CreateObject("Scripting.FileSystemObject")
    FSO.MoveFile "C:\dist\34.txt", "C:\temp\36.txt"
End Sub
```

书写上述代码时的感受就是不弹出成员，也没有任何语法提示。但是上述过程可以正常执行，实现文件的移动或重命名。

在实际编程过程中，前期绑定和后期绑定的代码通过改写，就可以转换，但是对于刚刚学习一个新对象库，推荐使用前期绑定方式。因为使用这种方式可以快速了解新对象的模型结构和语法特征。

1.3.3 FSO 对象模型

FSO 对象主要包括：Drive（分区、磁盘驱动器）、Folder（文件夹、路径）、File（文件）、TextStream（文本文件），以及 FileSystemObject 这五类对象。

1.3.4 遍历磁盘分区

FSO 对象模型中的 Drive 对象可以表达一个分区。FSO.Drives 是一个集合对象，用来返回所有分区。

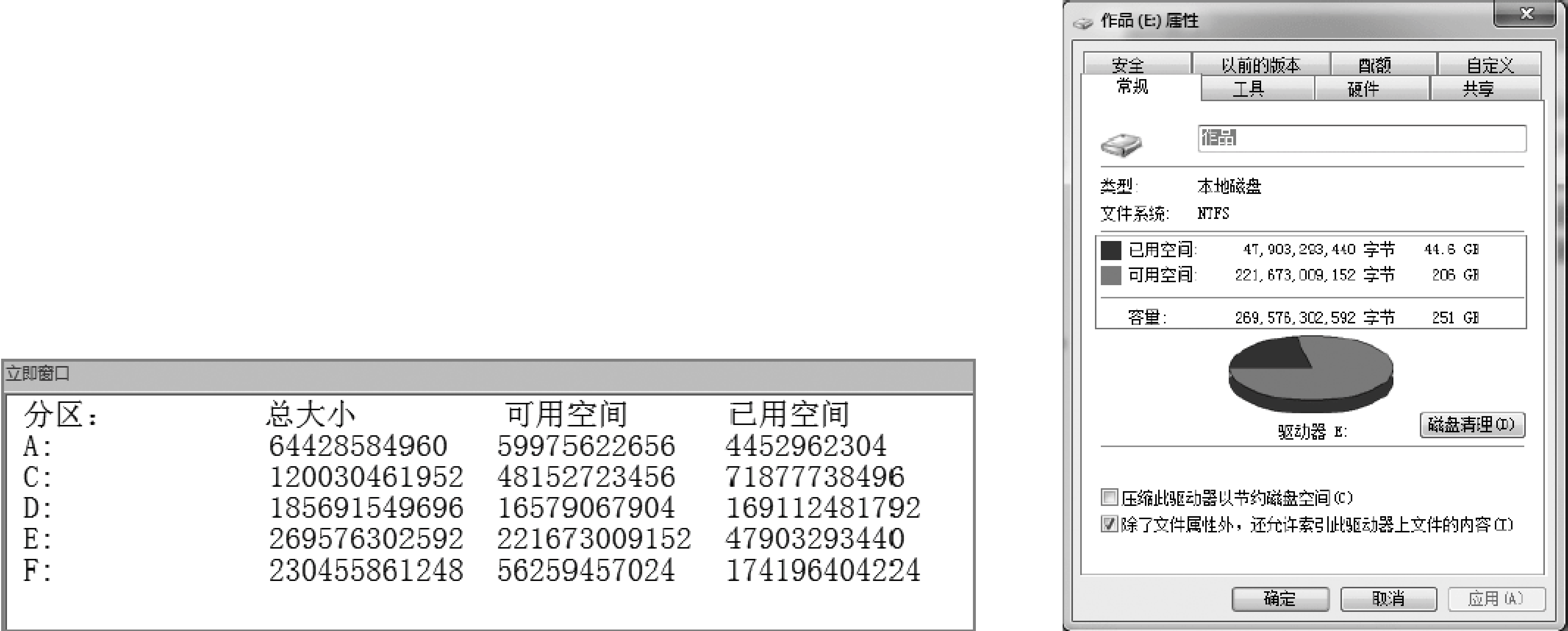
下面的过程遍历计算机的所有分区的名称、总大小、可用空间、已用空间。其中已用空间是用总大小减去可用空间得到的。

```
Sub 遍历磁盘分区()
    On Error Resume Next
    Dim fso As New Scripting.FileSystemObject
    Dim drv As Scripting.Drive
```

```
    MsgBox "分区总数：" & fso.Drives.Count
    Debug.Print "分区：", "总大小", "可用空间", "已用空间"
    For Each drv In fso.Drives
        With drv
            Debug.Print .Path, .TotalSize, .FreeSpace, .TotalSize - .FreeSpace '单位：字节
        End With
    Next drv
End Sub
```

运行上述过程，立即窗口的结果如图 1-24 所示。

在计算机的资源管理器中查看 E: 盘的属性，可以看到 E: 盘的总大小和可用空间与 VBA 运行结果是一致的，如图 1-25 所示。



立即窗口

分区：	总大小	可用空间	已用空间
A:	64428584960	59975622656	4452962304
C:	120030461952	48152723456	71877738496
D:	185691549696	16579067904	169112481792
E:	269576302592	221673009152	47903293440
F:	230455861248	56259457024	174196404224

图 1-24　遍历磁盘分区　　　图 1-25　核对磁盘分区大小

如果分区大小改写为更大容量单位的，首先要了解如下换算关系。

1GB=1024MB

1MB=1024KB

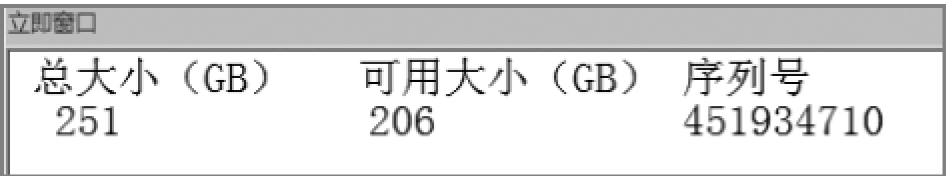
1KB=1024B，B 表示字节

可以看出 $1GB=1024^3B$

下面的过程单独查看 E: 盘的总大小和可用空间，用 GB 表示。

```
Sub 查看分区 ()
    Dim e As Scripting.Drive
    Dim fso As New Scripting.FileSystemObject
    Set e = fso.Drives.Item("E:")
    Debug.Print "总大小(GB)", "可用大小(GB)", "序列号"
    Debug.Print Int(e.TotalSize / 1024 ^ 3), Int(e.AvailableSpace / 1024 ^ 3),
e.SerialNumber
End Sub
```

上述程序的运行结果如图 1-26 所示。



立即窗口

总大小（GB）	可用大小（GB）	序列号
251	206	451934710

图 1-26　查看磁盘分区属性

1.3.5 操作文件夹

FSO 对象模型的 Folder 对象表示一个文件夹，或者称为一个路径。Folder 对象本身有大量的属性、成员和方法可以使用。

要表达一个文件夹，只能使用 FSO.GetFolder(" 文件夹路径 ") 的方式。

下面的过程查看一个文件夹的总大小、子文件夹的个数和文件的个数。

```
Sub 查看文件夹 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder
    Set fd = fso.GetFolder("C:\dist")
    With fd
        Debug.Print .Size, .SubFolders.Count, .Files.Count
    End With
End Sub
```

运行结果如图 1-27 所示。

在资源管理器中查看 dist 文件的属性，对比后发现文件夹的大小和运行结果是一致的。但是，文件夹和文件的个数不一样，这是因为 FSO 中的 SubFolders 和 Files 是文件夹直属的文件夹和文件，不包括子文件夹以及子文件夹中的文件。

资源管理器中看到的则是该文件夹中包含的所有文件夹和文件（包括递归嵌套的子文件夹），如图 1-28 所示。

立即窗口

173302736 2 23

图 1-27 查看文件夹的大小以及子文件夹和文件的数量

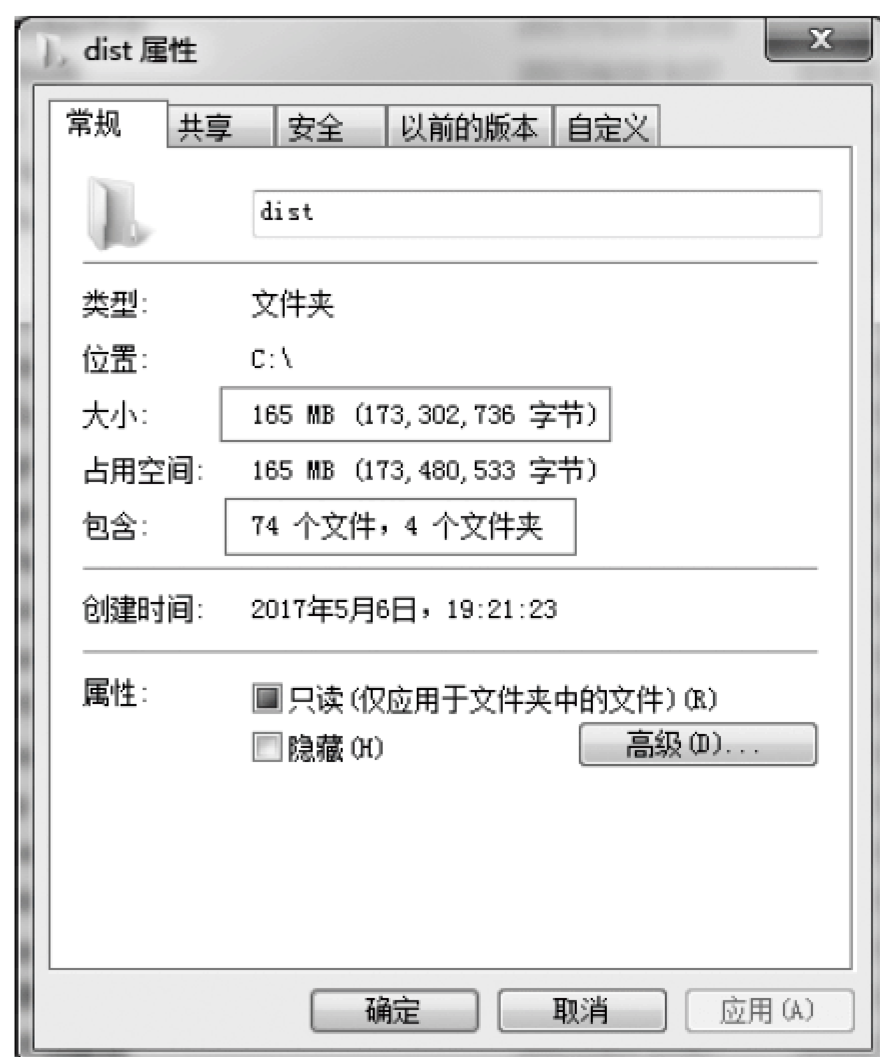


图 1-28 查看文件夹属性

下面的过程列出了文件夹 Folder 对象的其他常用属性。

```
Sub 文件夹的常用属性 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder
    Set fd = fso.GetFolder("C:\temp\Demo")
    With fd
```

```
        Debug.Print "文件夹属性", .Attributes
        Debug.Print "文件夹创建日期", .DateCreated
        Debug.Print "文件夹访问日期", .DateLastAccessed
        Debug.Print "文件夹修改日期", .DateLastModified
        Debug.Print "文件夹所属分区", .Drive
        Debug.Print "是否为根文件夹", .IsRootFolder
        Debug.Print "文件夹名称", .Name
        Debug.Print "父级文件夹", .ParentFolder.Name
        Debug.Print "文件夹路径", .Path
        Debug.Print "文件夹短名称", .ShortName
        Debug.Print "文件夹短路径", .ShortPath
        Debug.Print "文件夹大小", .Size
        Debug.Print "文件夹类型", .Type
    End With
End Sub
```

运行上述程序，立即窗口的结果如图 1-29 所示。

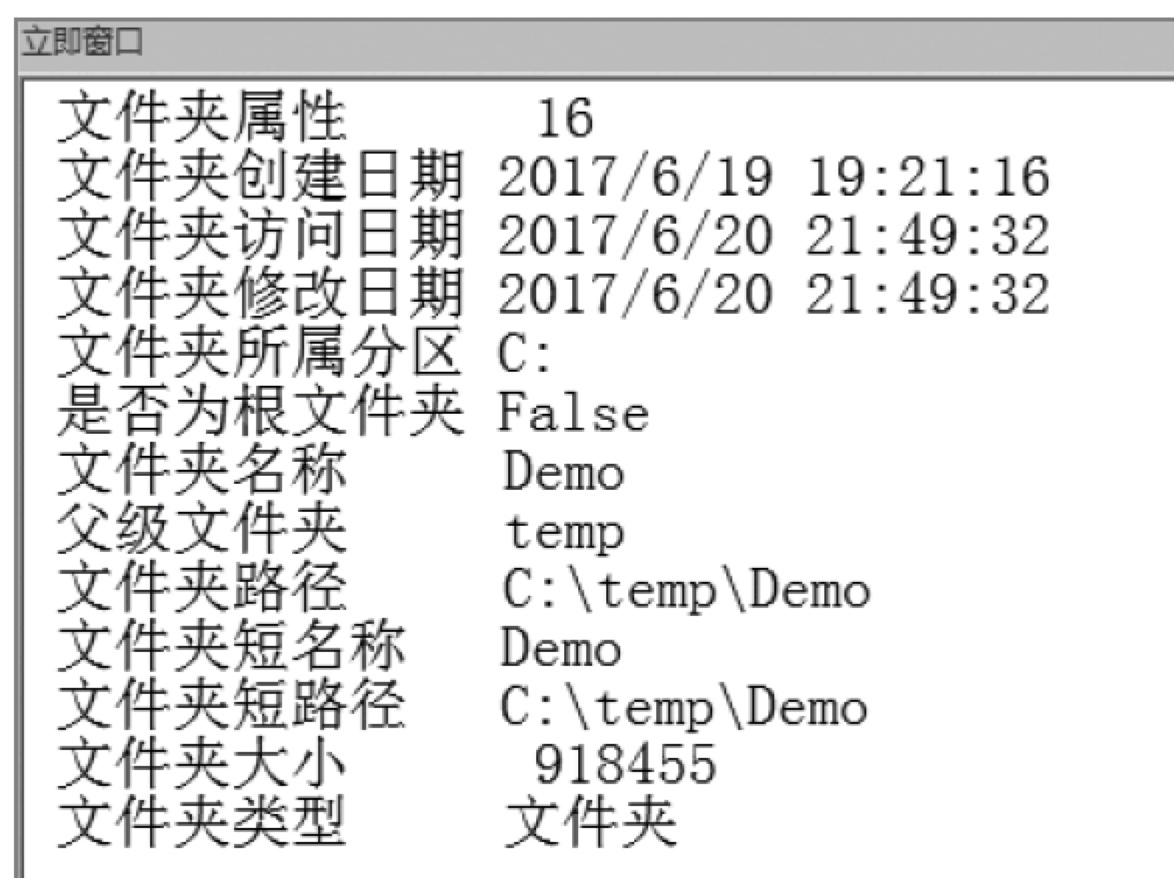

立即窗口
文件夹属性　　　16
文件夹创建日期　2017/6/19 19:21:16
文件夹访问日期　2017/6/20 21:49:32
文件夹修改日期　2017/6/20 21:49:32
文件夹所属分区　C:
是否为根文件夹　False
文件夹名称　　　Demo
父级文件夹　　　temp
文件夹路径　　　C:\temp\Demo
文件夹短名称　　Demo
文件夹短路径　　C:\temp\Demo
文件夹大小　　　918455
文件夹类型　　　文件夹

图 1-29　文件夹的有关属性

文件夹 Folder 对象的常用方法主要有 Copy、Move 和 Delete 等，用于复制、移动、删除文件夹。下面的过程首先创建一个空文件夹，然后重命名文件夹，最后删除该文件夹。

```
Sub 文件夹的常用方法()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder
    Set fd = fso.CreateFolder("C:\temp\2019")   '在 temp 下创建一个名为 2019 的文件夹
    fd.Move Destination:="C:\temp\2020" '把文件夹重命名为 2020
    fd.Delete Force:=True                  'Force 为 True 表示可以删除具有只读属性的文件夹
End Sub
```

需要注意的是，FSO 对象模型中文件夹的 Copy、Move、Delete 方法对于非空文件夹同样有效，也就是说，即使被操作的文件夹包含文件和子文件夹，也被一起复制、移动和删除。

要获取和返回一个文件夹 Folder 对象，除了上面介绍过的 GetFolder、CreateFolder 方法以外，还可以使用 Folder 对象的 SubFolders、ParentFolder 得到文件夹的子文件夹和父级文件夹。

1.3.6 文件夹拒绝访问的问题

磁盘根目录下除了包含正常的文件夹外，经常还包含一些隐藏的系统文件夹，当用 FSO 读写这些系统文件夹时，会弹出“拒绝访问”的错误。

如果计算机的文件夹选项中设置了不显示隐藏的文件和文件夹或驱动器，那么在资源管理器中看到的全是可以正常操作的文件夹，如图 1-30 所示。

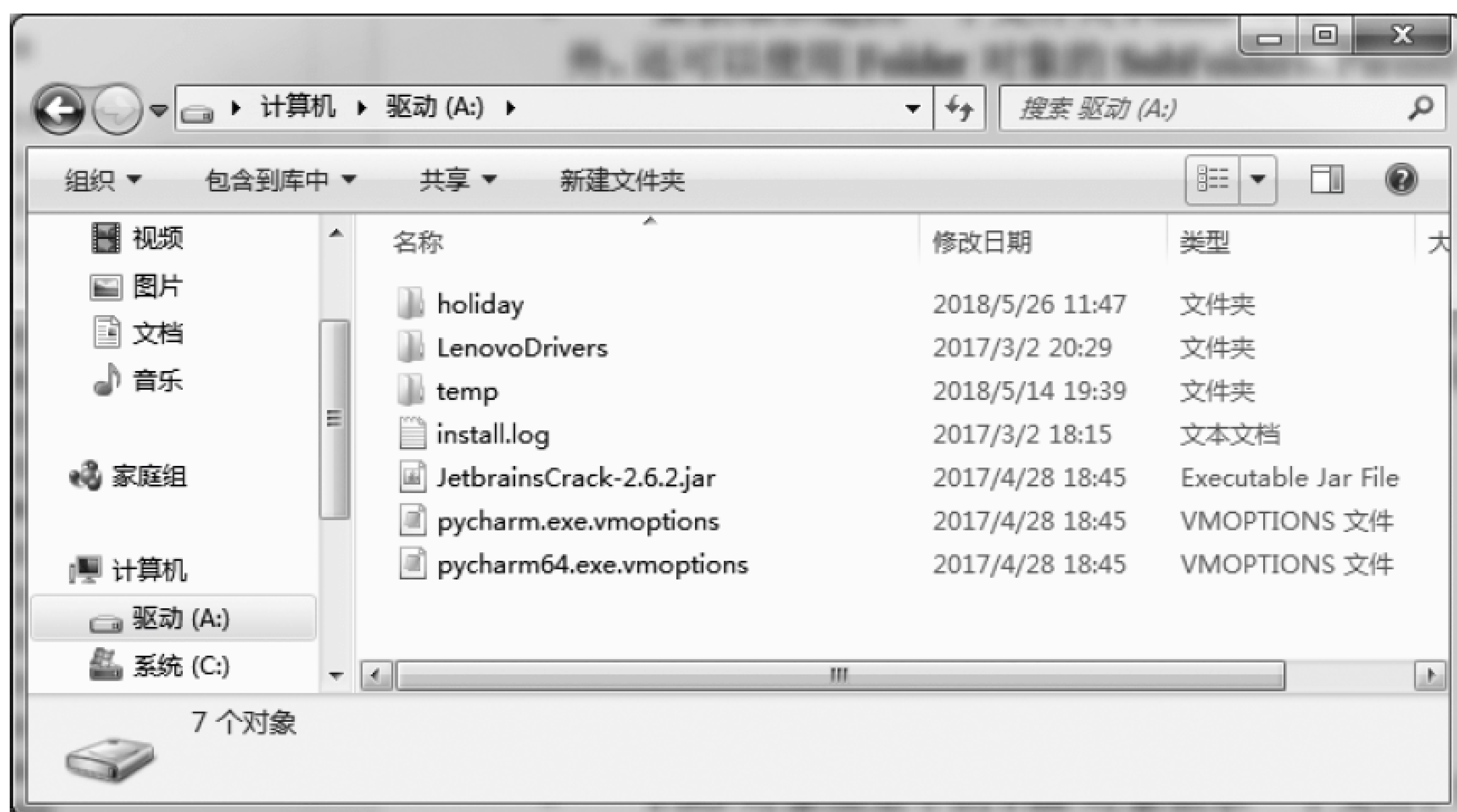


图 1-30 不显示隐藏的文件夹

通过更改“文件夹选项”，切换到“查看”选项卡，找到“隐藏受保护的操作系统文件”，去掉勾选，并且选择“显示隐藏的文件、文件夹和驱动器”，如图 1-31 所示。

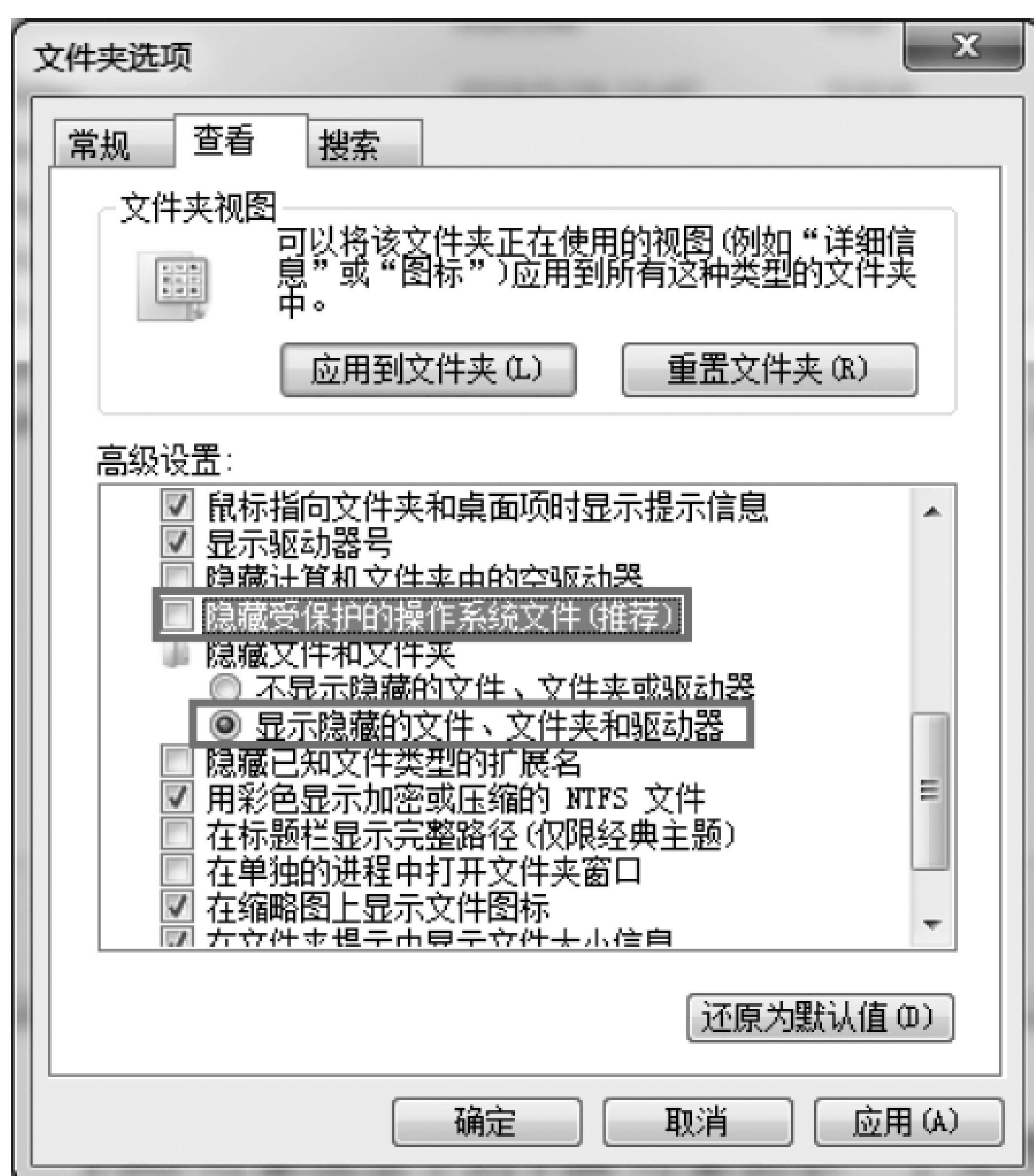


图 1-31 显示隐藏的文件、文件夹和驱动器

设置完毕后，A: 盘根目录下看到了隐藏的文件夹（图标比较虚），如图 1-32 所示。

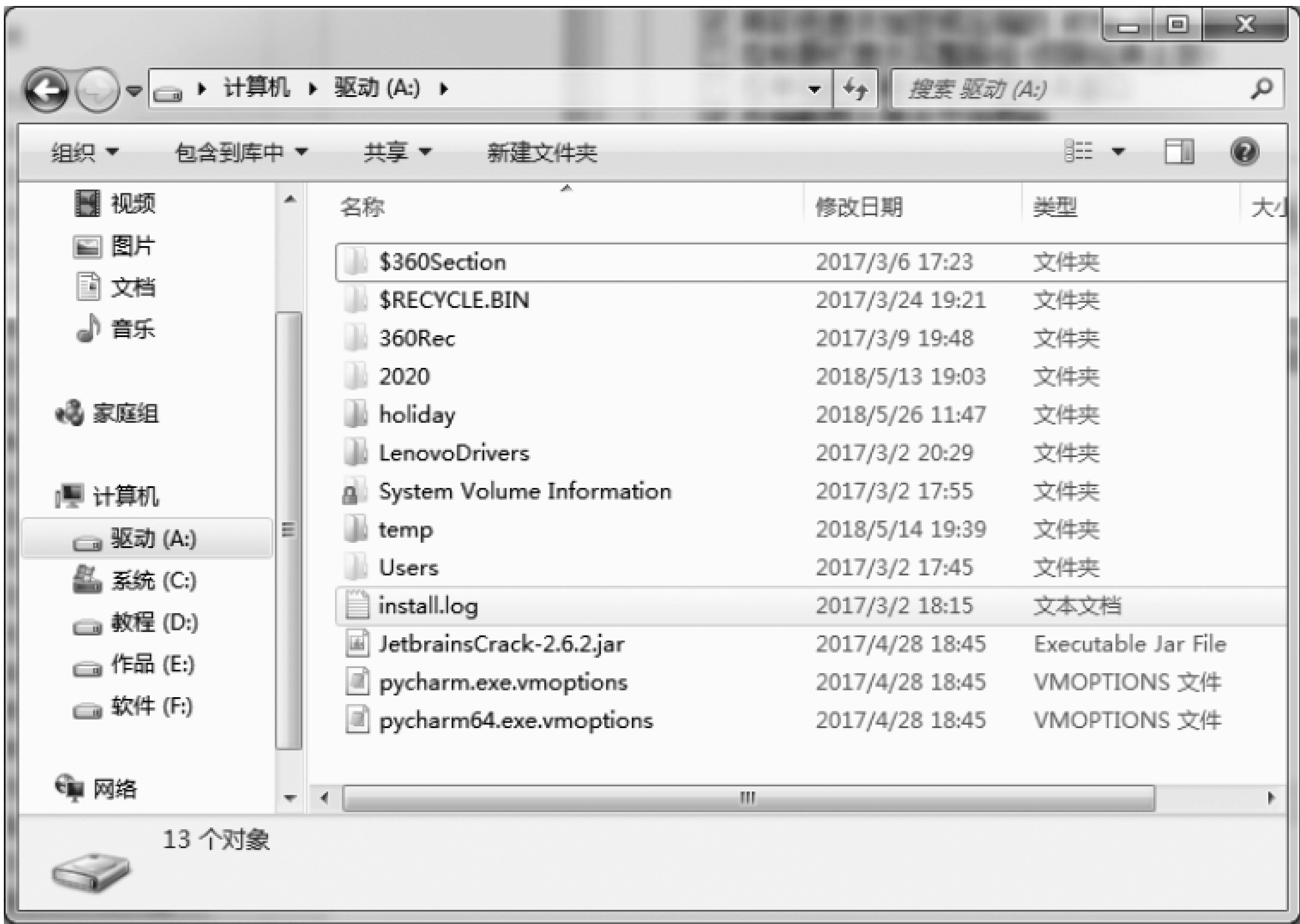


图 1-32　显示文件夹中隐藏的内容

这些隐藏文件夹大多数是系统文件夹，因此它们的属性是由 vbHidden+vbSystem+vbDirectory 组合的，结果为 22。正常文件夹的结果是 16。

下面的程序遍历 A: 盘根目录下的所有子文件夹，如果不是隐藏文件夹，就打印其名称、包含的子文件夹个数、属性值。

```
Sub 处理文件夹拒绝访问 ()
    Dim FSO As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder
    For Each fd In FSO.GetFolder("A:").SubFolders
        If (fd.Attributes And vbHidden) <> vbHidden Then
            Debug.Print fd.Name, fd.SubFolders.Count, fd.Attributes
        End If
    Next fd
End Sub
```

以上代码中，If 判断语句起到过滤文件夹的作用，不处理隐藏的文件夹。

运行上述程序，立即窗口的打印结果如图 1-33 所示。

可以看出，只有 3 个文件夹是正常文件夹。

假设去掉上述代码中的 If 判断语句，再次运行上述程序，当遍历到“System Volume Information”这个文件夹，访问 fd.SubFolders.Count 属性时，弹出如图 1-34 所示运行时错误。

立即窗口

```
holiday        0        16
LenovoDrivers  1        16
temp           6        16
```

图 1-33　只列举正常的文件夹

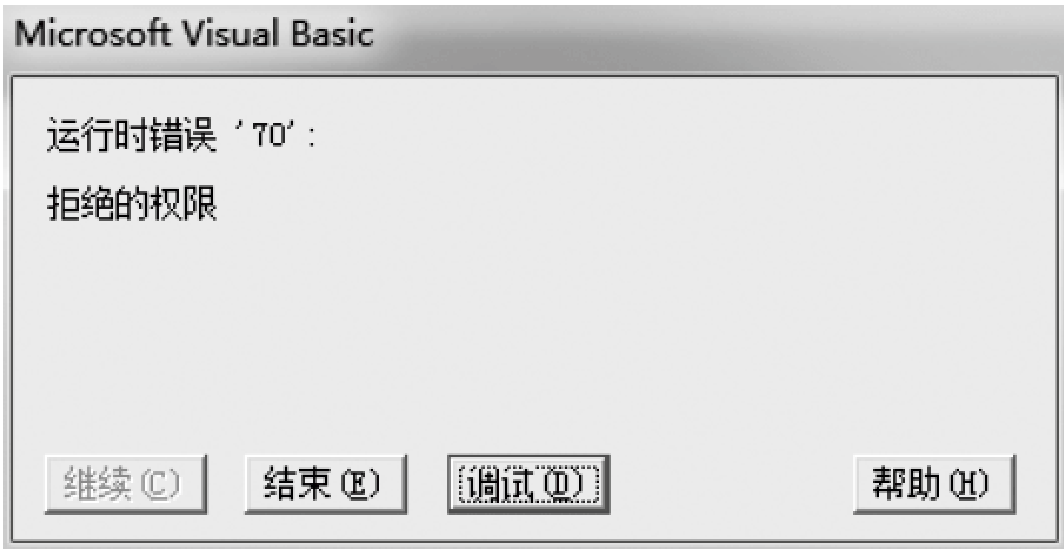


图 1-34　不可访问系统文件夹

综上所述，在处理文件夹时，需要考虑到该文件夹能否被访问，必要时需要补充上述过滤条件。

1.3.7 操作文件

FSO 对象模型中的 File 对象表示一个文件。与 Folder 对象类似，File 对象也有很多的属性和方法。

在下面的过程中，用 GetFile 方法获取一个文件后，遍历该文件的常用属性。

```
Sub 文件的常用属性 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fl As Scripting.File
    Set fl = fso.GetFile("C:\temp\abc.xls")
    With fl
        Debug.Print "文件属性", .Attributes
        Debug.Print "文件创建日期", .DateCreated
        Debug.Print "文件访问日期", .DateLastAccessed
        Debug.Print "文件修改日期", .DateLastModified
        Debug.Print "文件所属分区", .Drive
        Debug.Print "文件名称", .Name
        Debug.Print "父级", .ParentFolder.Name
        Debug.Print "文件路径", .Path
        Debug.Print "文件短名称", .ShortName
        Debug.Print "文件短路径", .ShortPath
        Debug.Print "文件大小", .Size
        Debug.Print "文件类型", .Type
    End With
End Sub
```

上述程序的运行结果如图 1-35 所示。

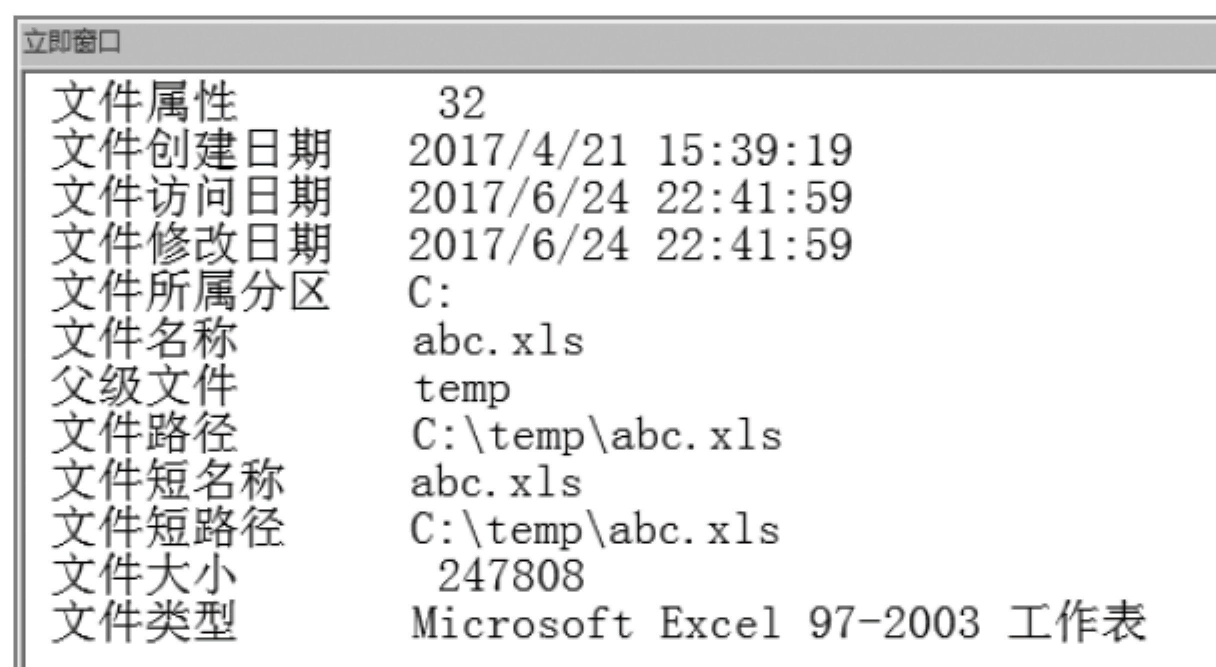


图 1-35 文件的属性

文件对象 File 的常用方法有 Copy、Move、Delete。下面通过一段代码进行了解。

```
Sub 文件的常用方法 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fl As Scripting.File
    Set fl = fso.GetFile("C:\temp\abc.xls")
    fl.Copy Destination:="C:\dist\xyz.xls"                '复制文件
```

```
    Set fl = fso.GetFile("C:\dist\xyz.xls")
    fl.Delete                                        '删除文件
    Set fl = fso.GetFile("C:\temp\abc.xls")
    fl.Move Destination:="C:\temp\123.xls"           '移动或重命名文件
End Sub
```

代码分析：先把 abc.xls 复制到 dist 文件夹下，并修改名称为 xyz.xls。接着删除 xyz.xls，最后把 abc.xls 重命名为 123.xls。

1.3.8　遍历文件

文件夹对象下面有一个 Files 集合对象，表示该文件夹下的所有文件。可以据此来遍历文件夹下的直属文件。该方法无法遍历包含在子文件夹中的文件。

下面的程序遍历 CTEX 文件夹下的所有文件，并且打印每个文件的重要属性。

```
Sub 遍历所有文件()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder, fl As Scripting.File
    Set fd = fso.GetFolder("C:\CTEX")
    Debug.Print "名称", "修改日期", "类型", "大小"
    For Each fl In fd.Files
        Debug.Print fl.Name, fl.DateLastModified, fl.Type, fl.Size / 1024 & " KB"
    Next fl
End Sub
```

上述程序的运行结果如图 1-36 所示。

```
立即窗口
名称          修改日期              类型        大小
Changes.txt   2012/3/22 16:16:12    文本文档    11.33984375 KB
Readme.txt    2012/3/22 12:50:36    文本文档    3.580078125 KB
Repair.exe    2012/3/22 16:16:22    应用程序    126.3466796875 KB
Uninstall.exe 2017/6/6 8:43:48      应用程序    86.2841796875 KB
```

图 1-36　遍历文件夹中的所有文件

在资源管理器中查看文件的属性，发现和输出结果一致，如图 1-37 所示。

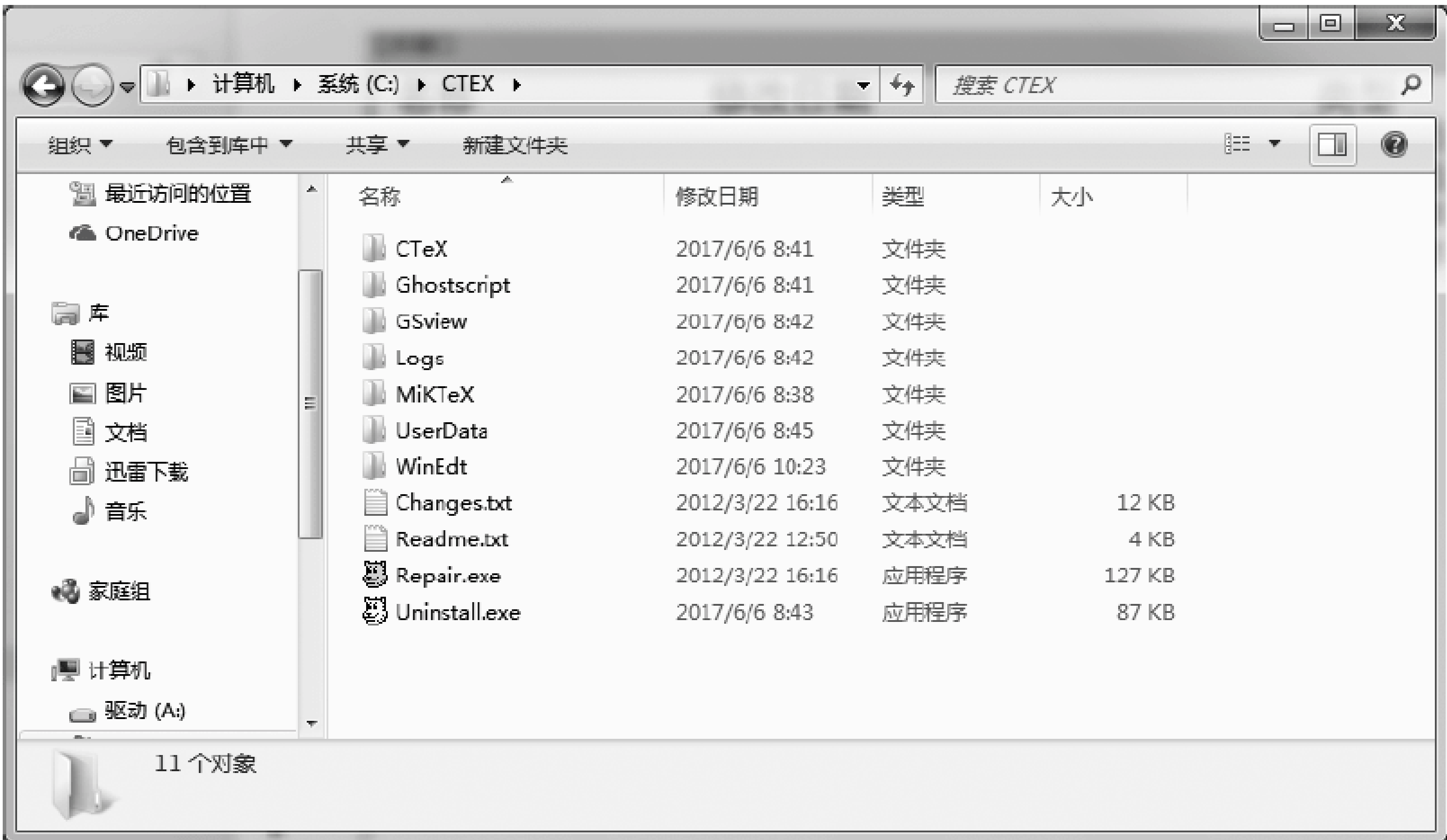


图 1-37　核对文件属性

如果要选择性地遍历文件，例如只遍历文件夹中的文本文件，只需要在 For 循环中嵌套 If 语句，判断一下扩展名即可。

```
Sub 遍历文本文件 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder, fl As Scripting.File
    Set fd = fso.GetFolder("C:\CTEX")
    For Each fl In fd.Files
        If fso.GetExtensionName(fl.Path) = "txt" Then
            Debug.Print fl.Path
        End If
    Next fl
End Sub
```

代码分析：FSO 的 GetExtensionName 可以返回指定文件的扩展名。

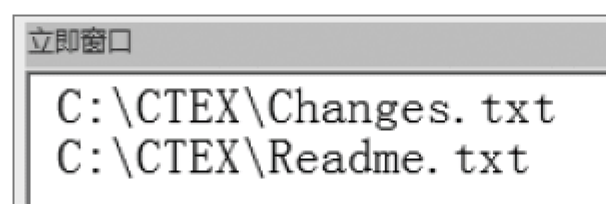


图 1-38　只遍历指定扩展名的文件

上述程序的运行结果如图 1-38 所示。

需要注意的是：在用 For Each 循环遍历文件夹中的所有文件时，尽量不要在遍历的同时对文件进行重命名、删除、复制、移动等操作，以免发生不可预料的结果。比较安全的做法是遍历的时候可以先把所有文件名存储到数组或字典中，后期对数组或字典进行操作。

1.3.9　遍历子文件夹

一个文件夹中可能包含多个子文件夹，在 FSO 对象模型中，SubFolders 集合对象表示磁盘分区或者文件夹下面的所有子文件夹。

下面的代码遍历 CTEX 文件夹下的所有子文件夹，打印每个子文件夹的路径，以及子文件夹中包含的文件总数。

```
Sub 遍历子文件夹 ()
    Dim fso As New Scripting.FileSystemObject
    Dim fd As Scripting.Folder, fds As Scripting.Folder
    Set fd = fso.GetFolder("C:\CTEX")
    MsgBox "子文件夹个数为: " & fd.SubFolders.Count
    Debug.Print "文件夹路径", "文件总数"
    For Each fds In fd.SubFolders
        Debug.Print fds.Path, fds.Files.Count
    Next fds
End Sub
```

上述程序的运行结果如图 1-39 所示。

立即窗口

文件夹路径	文件总数
C:\CTEX\CTeX	0
C:\CTEX\Ghostscript	0
C:\CTEX\GSview	0
C:\CTEX\Logs	11
C:\CTEX\MiKTeX	0
C:\CTEX\UserData	0
C:\CTEX\WinEdt	15

图 1-39　每个文件夹包含的内容

实际上，Windows 的文件、路径管理是一个树状结构，文件夹中可以包含子文件夹和文件，子文件夹也是一种文件夹，其中还能包含子文件夹和文件，从逻辑上讲，子文件夹可以无限层嵌套。

如果要遍历到某位置下的所有子文件夹和文件，需要用递归算法反复访问 SubFolders 才能实现。

本书源代码文件“实例文档 02.xlsm”中的 UserForm1 使用了 Treeview 控件结合递归算法来展示文件夹中的所有子文件夹和文件，如图 1-40 所示。

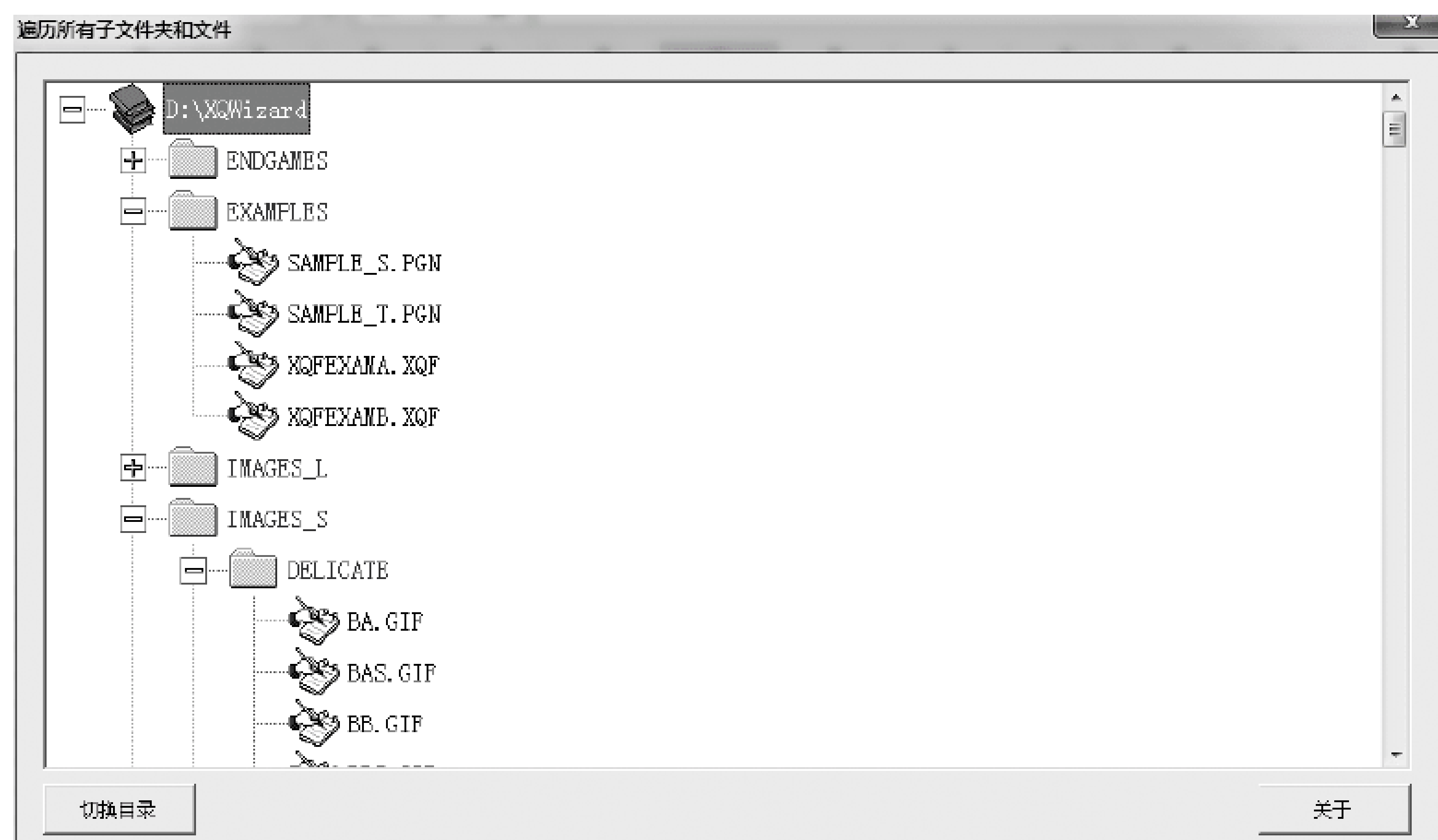


图 1-40　递归遍历文件管理系统

读者可以下载源代码文件自行研究。

下面的代码使用递归算法遍历任意路径，并且把遍历的结果发送到 Excel 单元格中。

```
Public Root As Folder, fd As Folder, fl As File
Public Record As Long
Private Sub Traversal()
    Dim FSO As New FileSystemObject
    Set Root = FSO.GetFolder("E:\ExcelObject_VSTO_VBA")
    ActiveSheet.UsedRange.ClearContents
    Record = 0
    Recursion Root, 0
End Sub
Private Sub Recursion(ByVal CurrentFolder As Folder, ByVal Layer As Integer)
    Layer = Layer + 1
    Record = Record + 1
    ActiveSheet.Cells(Record, Layer).Value = CurrentFolder.Name
    For Each fl In CurrentFolder.Files
        Record = Record + 1
        ActiveSheet.Cells(Record, Layer + 1).Value = fl.Name
    Next fl
    For Each fd In CurrentFolder.SubFolders
        If fd.SubFolders.Count + fd.Files.Count > 0 Then
            Recursion fd, Layer
        End If
    Next fd
End Sub
```

运行代码中的 Traversal 过程，从根目录“E:\ExcelObject_VSTO_VBA”反复调用 Recursion 过程，如图 1-41 所示。

A1 ExcelObject_VSTO_VBA

	A	B	C	D	E	F	G	H	I	J
1	ExcelObject_VSTO_VBA									
2		ExcelObject_VSTO_VBA.sln								
3		ExcelObject_VSTO_VBA.v11.suo								
4		ExcelObject_VSTO_VBA								
5			ExcelObject_VSTO_VBA.csproj							
6			ExcelObject_VSTO_VBA_TemporaryKey.pfx							
7			Resource1.Designer.cs							
8			Resource1.resx							
9			Ribbon1.cs							
10			Share.cs							
11			ThisAddIn.cs							
12			ThisAddIn.Designer.cs							
13			ThisAddIn.Designer.xml							
14			UserControl1.cs							
15			UserControl1.Designer.cs							
16			UserControl1.resx							
17			bin							
18				Debug						
19					CodeLibrary.mdb					
20					ExcelObject_VSTO_VBA.dll					
21					ExcelObject_VSTO_VBA.dll.manifest					
22					ExcelObject_VSTO_VBA.pdb					
23					ExcelObject_VSTO_VBA.vsto					
24					Help.chm					
25					Microsoft.Office.Tools.Common.v4.0.Utilities.dll					
26					Microsoft.Office.Tools.Common.v4.0.Utilities.xml					
27				Release						
28					1.1 本工具的运行环境.htm					
29					1.2 工具界面简介.htm					
30					2.1 通过点击功能区按钮显示代码.htm					
31					2.2 点击树形结构显示代码.htm					

Sheet1

图 1-41　递归遍历的结果发送到单元格

上述代码的源文件为“实例文档 03.xlsm”。

1.3.10　FSO 的更多操作方式

前面介绍过的文件、文件夹的操作（Copy、Move、Delete）是以文件 / 文件夹对象为主体的。针对这种方式，一行代码只能操作一个文件或路径。

FSO 允许以 FSO 对象作为主体，这种情形下，以通配符作为参数，从而达到一行代码就可以批处理文件或路径。常用操作有以下 6 个：

- FSO.CopyFile
- FSO.CopyFolder
- FSO.MoveFile
- FSO.MoveFoler
- FSO.DeleteFile
- FSO.DeleteFoler

文件或路径中的通配符可以使用 * 来匹配任意多个字符，也可以使用 ? 匹配任意一个字符。

假设 C:\temp 下面有大量的文件夹，下面的过程可以把 p 开头的所有文件夹一次性删除。

```
Sub Test1()
    Dim FSO As New FileSystemObject
```

```
    FSO.DeleteFolder "C:\temp\p*"
End Sub
```

运行上述过程，以 p 开头的文件夹就被删除了，如图 1-42 所示。

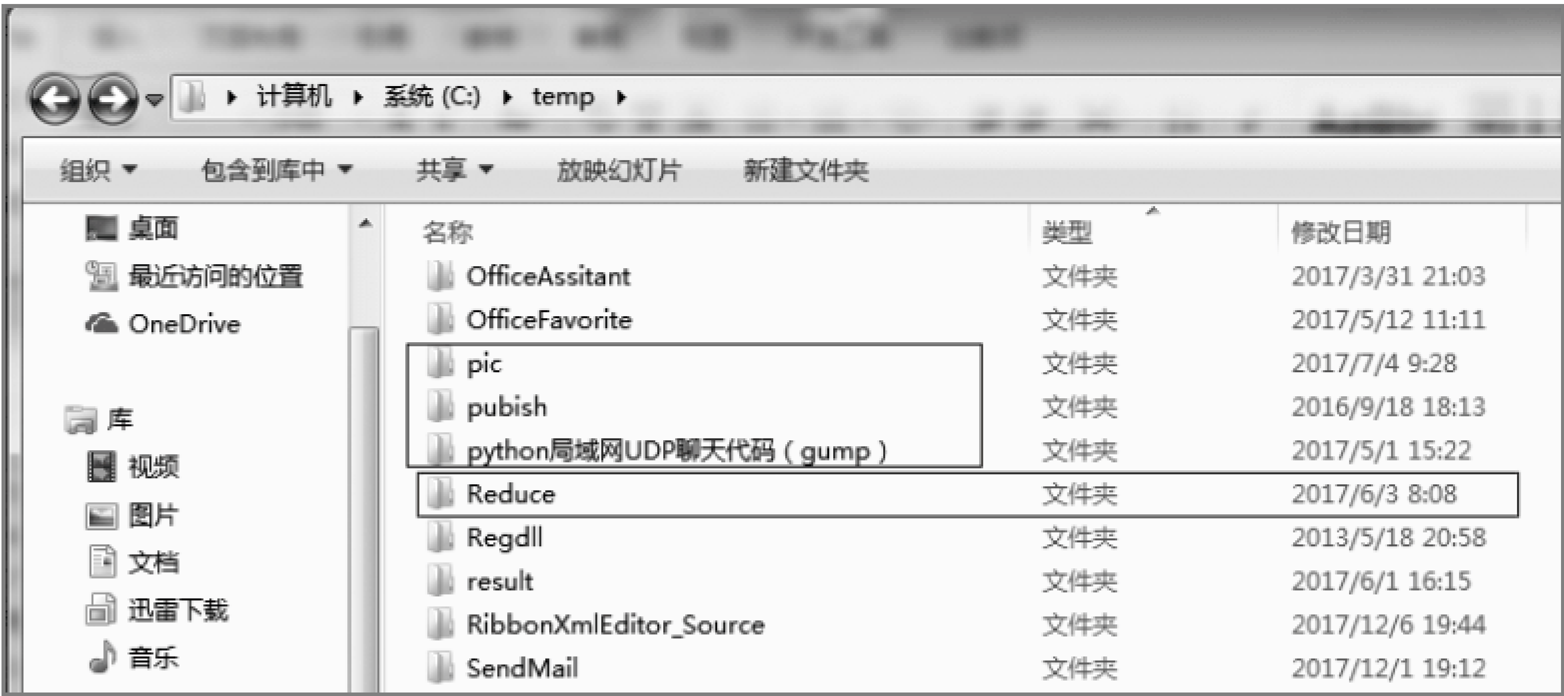


图 1-42　用通配符限定被处理的文件夹

下面再举一个批量移动文件的实例。datas 文件夹下有大量的记事本文件，下面的代码可以把一位数命名的文件批量转移到 2018 文件夹中。

```
Sub Test2()
    Dim FSO As New FileSystemObject
    FSO.MoveFile "C:\temp\datas\?.txt", "C:\temp\2018\"
End Sub
```

代码分析：?.txt 只能匹配到 9.txt 等，但是不能匹配 12.txt，也就是说 ? 只能代表一个字符。因此，运行上述过程后，图中框内的文件被批量转移，如图 1-43 所示。

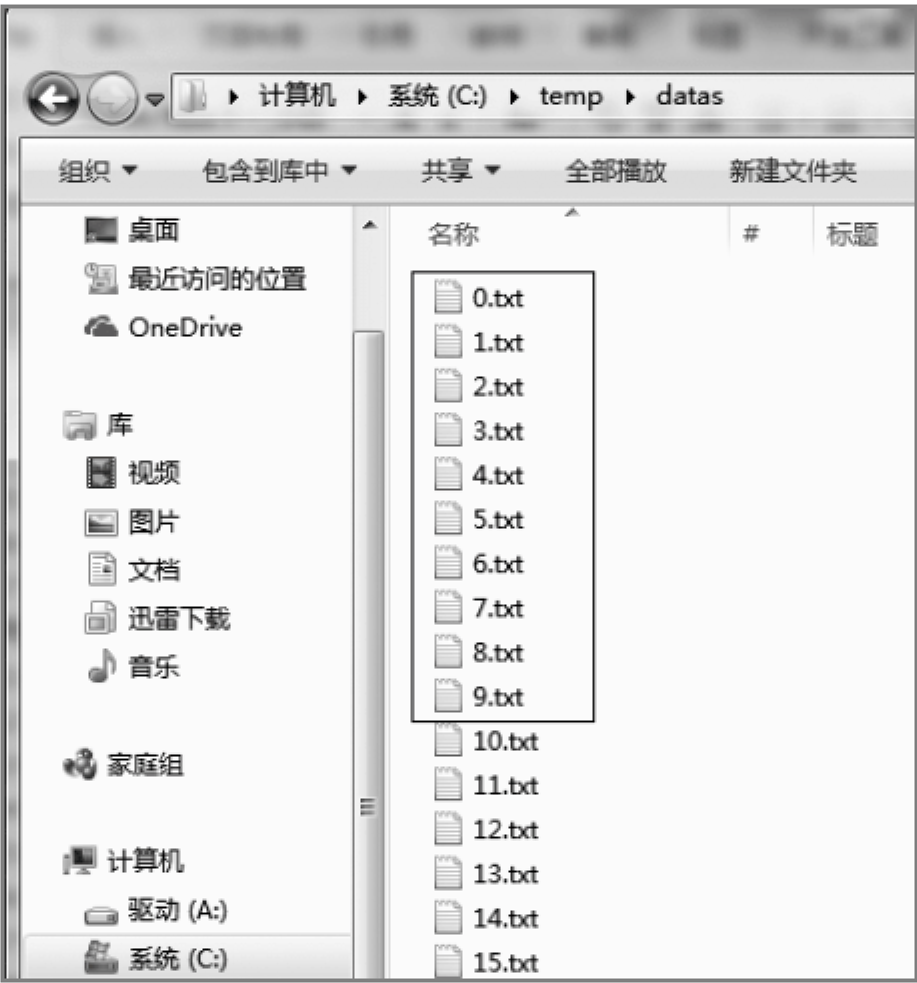


图 1-43　用通配符限定文件

此外，使用 FSO 还可以快速获取计算机中的特殊文件夹。

```
Sub Test3()
    Dim FSO As New FileSystemObject
    Debug.Print FSO.GetSpecialFolder(0), "操作系统文件夹"
    Debug.Print FSO.GetSpecialFolder(1), "System32 文件夹"
```

```
    Debug.Print FSO.GetSpecialFolder(2), "临时文件夹"
End Sub
```

上述程序的运行结果如图 1-44 所示。



图 1-44 获取特殊文件夹

1.3.11 判断是否存在

在利用 FSO 对计算机中的磁盘分区、文件夹、文件进行操作时，必须事先确保目标存在方可进行操作，因此，FSO 提供了 DriveExists、FolderExists、FileExists 三个函数来快速判断目标是否存在，这三个函数都返回布尔值，如果存在则返回 True。

以下三行代码分别判断计算机中是否存在 K 盘、文件夹 C:\temp，以及是否存在 Test.Spec 文件。

```
Sub Test4()
    Dim FSO As New FileSystemObject
    Debug.Print FSO.DriveExists("K:")
    Debug.Print FSO.FolderExists("C:\temp\")
    Debug.Print FSO.FileExists("D:\Test.spec")
End Sub
```

下面的代码先判断 C: 盘下是否有 Download 文件夹，如果没有，则创建这个文件夹。

```
Sub Test5()
    Dim FSO As New FileSystemObject
    If FSO.FolderExists("C:\Download\") = False Then
        FSO.CreateFolder "C:\Download\"
    End If
End Sub
```

在日常办公中，经常需要在成千上万个文件中核对哪些文件存在，哪些文件没有，现在假设 C:\Example 文件夹下有大量的压缩包文件，理论上从 1 月 1 日到 1月 20 日的文件名都有。现在需要核对哪些日期的文件不存在，如图 1-45 所示。

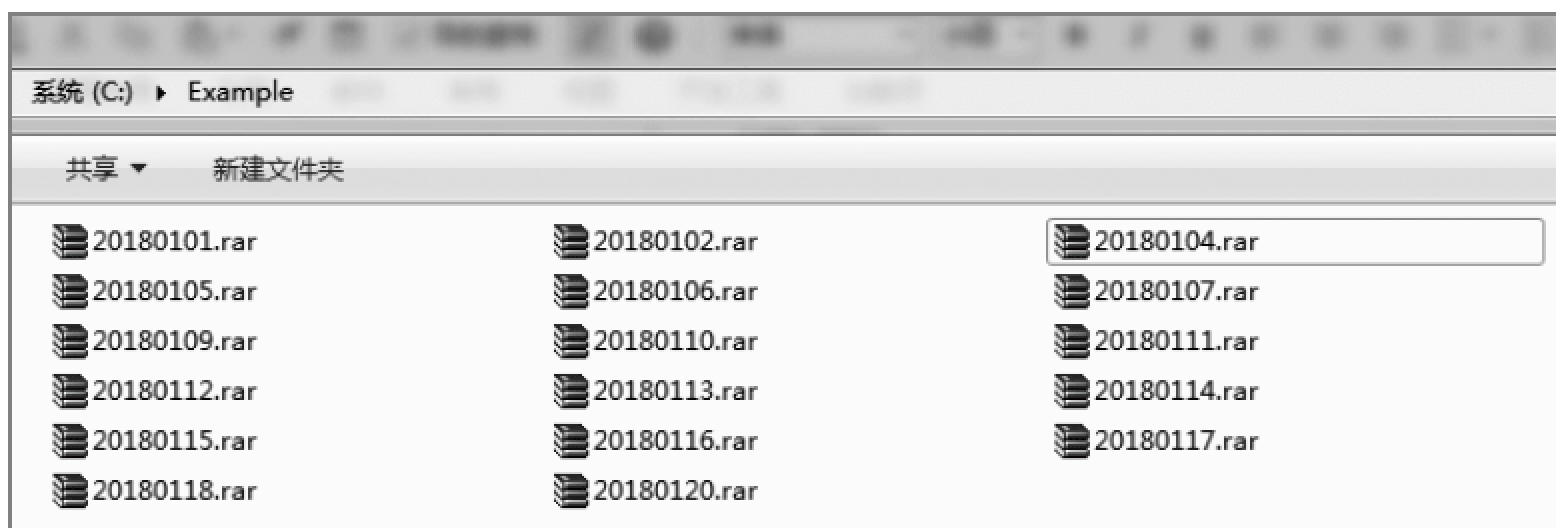


图 1-45 文件夹中的内容

下面的过程，在日期中循环，通过变化的日期产生临时的文件名，然后用 FSO 的 FileExists 来判断该日期对应的文件是否存在，不存在的话就打印到立即窗口。

```
Sub 检查文件是否存在 ()
    Dim FSO As New FileSystemObject
    Dim i As Date
    Dim parent As String
    Dim fname As String
    parent = "C:\Example\"
    For i = #1/1/2018# To #1/20/2018#
        fname = Format(i, "yyyymmdd") & ".rar"
        If FSO.FileExists(parent & fname) = False Then
            Debug.Print fname, " 不存在！ "
        End If
    Next i
End Sub
```

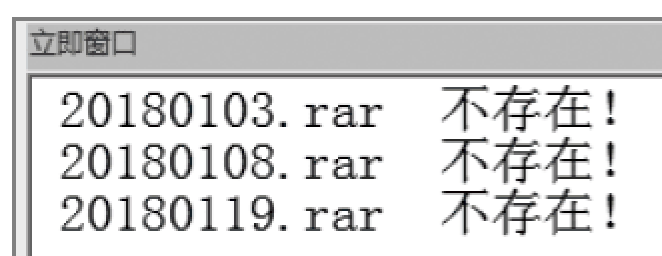


图 1-46　检查哪些名称的文件夹不存在

运行上述过程，文件夹中不存在的文件名打印在立即窗口中，如图 1-46 所示。

1.3.12　文本文件的读写

计算机中的文件大体可以分为文本文件和二进制文件两大类，文本文件可以用 Windows 自带的记事本软件打开，二进制文件（图片、Word 文档）则不能用记事本打开。

在编程过程中，经常需要对文本文件进行读写。下面介绍利用 FSO 中的 TextStream 对象操作文本文件。

对文件的读写操作分为以下三个环节。

- 打开文件（OpenTextStream）。
- 读 / 写（Read、Write、Append）。
- 关闭文件（Close）。

1. 打开文件

OpenTextStream 函数返回一个 TextStream 文件对象，该函数有以下 4 个参数。

- FileName：文件路径。
- IOMode：读写模式，有 ForReading、ForWriting、ForAppending 三种模式，含义分别是读取、写入和追加写入。
- Create：当文件不存在时，询问是否创建。该参数默认值为 False。
- Format：编码格式，取值必须是以下三者之一，TristateFalse（以 ANSI 格式打开）、TristateTrue（以 Unicode 格式打开）、TristateUseDefault（使用系统默认打开）。

2. 写入文件

TextStream 对象写入文件的方法有如下三种。

- Write：当前位置写入一个字符串。
- WriteLine：当前位置写入一个字符串，并在后面自动加一个换行符。

❑ WriteBlankLines：写入多个空行。

以下代码向记事本文件中自动写入一些内容。

```
Sub WriteMethod()
    Dim FSO As New FileSystemObject
    Dim txt As TextStream
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForWriting,
Create:=True, Format:=TristateFalse)
    With txt
        .Write "春雨惊春清谷天"
        .WriteLine "夏满芒夏暑相连"
        .WriteBlankLines 3
        .Write "冬雪雪冬小大寒"
        .Close
    End With
End Sub
```

代码分析：以上代码用 ANSI 格式打开 new.txt，如果路径下不存在该记事本文件，则会自动创建一个空文件。

Write 方法写入内容后，继续写入的内容会紧跟其后写入。代码中的 WriteBlankLines 3 表示输入三个换行符。写入操作完毕后，别忘记用 Close 方法关闭文件。

上述程序的运行结果如图 1-47 所示。

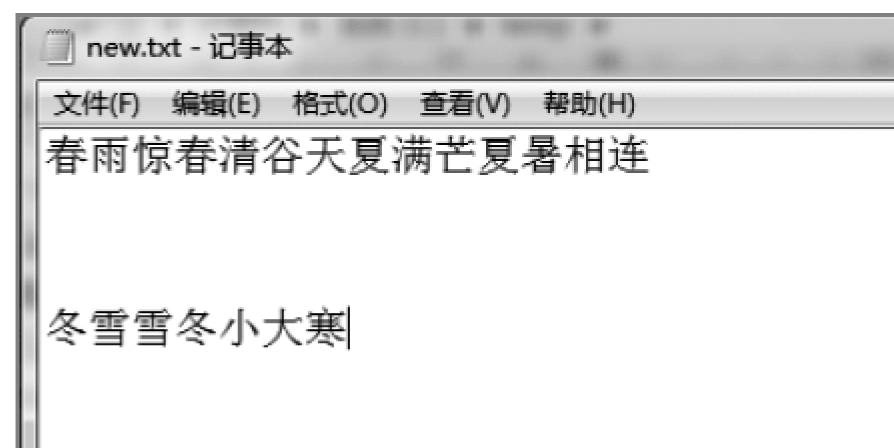


图 1-47　写入文本文件

3. 读取文件

TextStream 对象用于读取文本文件内容的方法有如下三种。

❑ Read(i)：在当前位置读取 i 个字符。

❑ ReadLine：读取一整行。

❑ ReadAll：读取整个文件内容。

当一个文本文件被打开时，指针处于文本内容的起始位置，读取过程中，指针（当前位置）随之向右移动。

假定 new.txt 文件中有一首完整的二十四节气歌，如图 1-48 所示。

下面用 Read 和 ReadLine 方法来读取一部分内容，赋给过程中的变量。

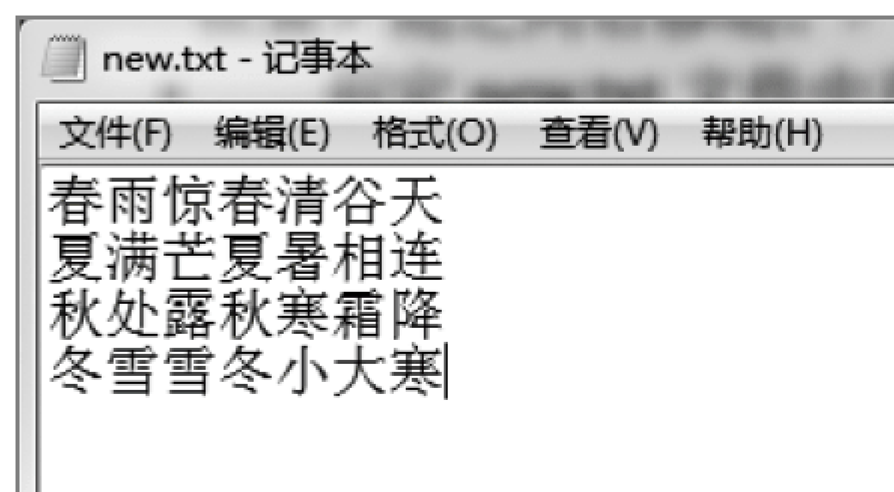


图 1-48　文本文件的内容

```
Sub ReadMethod()
    Dim FSO As New FileSystemObject
    Dim txt As TextStream
    Dim result(1 To 4) As String
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForReading,
Format:=TristateFalse)
    With txt
```

```
        result(1) = .Read(5)
        result(2) = .Read(5)
        result(3) = .ReadLine
        .Close
    End With
    Debug.Print result(1)
    Debug.Print result(2)
    Debug.Print result(3)
End Sub
```

代码分析：文件打开后，指针位于“春”之前，result(1) 读取 5 个字符，指针移动到“清”之后，result(2) 从当前位置再读取 5 个字符，也就是读取“谷天”、回车符（vbCr）、换行符（vbLf）、“夏”，总计 5 个，如图 1-49 所示。

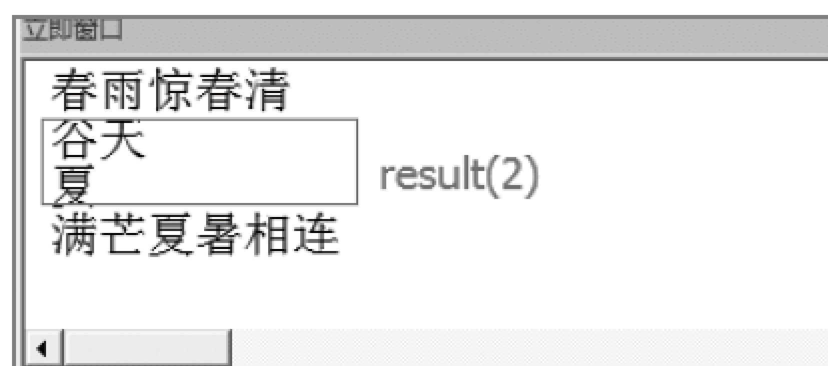


图 1-49　文件读取方法

result(3) 从当前位置读取到行尾。

在实际编程过程中，经常需要把记事本中的多行文本按行读取，发送到单元格中，或者列表框控件等，此时使用 ReadLine 是最好的选择。

如果要一次性读取所有内容，发送给文本框控件，用 ReadAll 最省事。

下面两个过程分别采用 ReadLine、ReadAll 方法从文本文件中读取内容，发送到列表框、文本框中。

```
Dim FSO As New FileSystemObject
Private Sub CommandButton1_Click()
    Dim txt As TextStream
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForReading,
Format:=TristateFalse)
    With txt
        Me.ListBox1.Clear
        Do Until .AtEndOfStream
            Me.ListBox1.AddItem .ReadLine
        Loop
        .Close
    End With
End Sub

Private Sub CommandButton2_Click()
    Dim txt As TextStream
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForReading,
Format:=TristateFalse)
    With txt
        Me.TextBox1.Text = ""
        Me.TextBox1.Text = .ReadAll
        .Close
    End With
End Sub
```

代码分析：在使用 TextStream 对象的 Read 以及 ReadLine 方法时，一定要判断是否已经读取到文件末尾。当读取到文件末尾时，TextStream 对象的 AtEndOfStream 属性会返回 True，因此经常利用该属性配合 Do 循环读取所有内容。

上述程序中，左侧是一个列表框控件，右侧是一个文本框（MultiLine 为 True）。

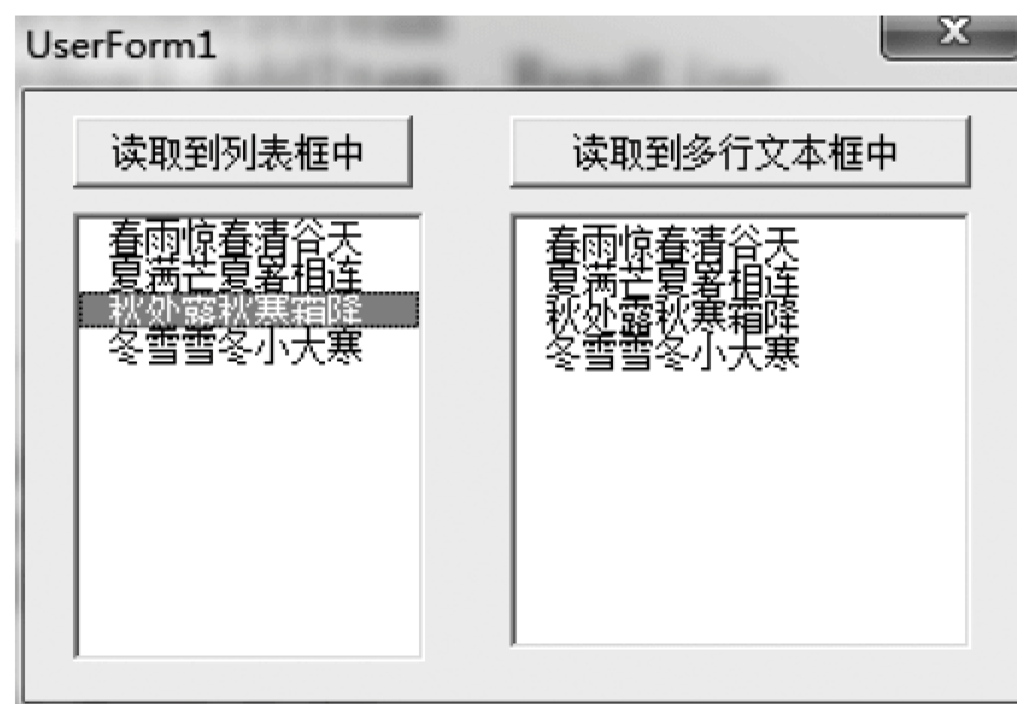


图 1-50　从文本文件读取内容到控件

上述程序的运行结果如图 1-50 所示。

读取文件的过程中，还可以使用 Skip 或 SkipLine 方法跳过字符或跳过行，相当于主动改变指针位置。

还是以二十四节气歌为例，理解一下 Skip 的作用。

```
Sub SkipMethod()
    Dim FSO As New FileSystemObject
    Dim txt As TextStream
    Dim result(1 To 4) As String
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForReading,
Format:=TristateFalse)
    With txt
        result(1) = .Read(2)
        .Skip 2
        result(2) = .Read(2)
        .Close
    End With
    Debug.Print result(1)
    Debug.Print result(2)
End Sub
```

代码分析：打开文件后，result(1) 首先读取前 2 个字符“春雨”，Skip 2 表示跳过 2 个字符（“惊春”两个字被跳过），接着 result(2) 读取两个字符，也就是读取“清谷”两个字。

下面的代码演示了如何跳过整行。

```
Sub SkipLineMethod()
    Dim FSO As New FileSystemObject
    Dim txt As TextStream
    Dim result(1 To 4) As String
    Set txt = FSO.OpenTextFile(Filename:="C:\temp\new.txt", IOMode:=ForReading,
Format:=TristateFalse)
    With txt
        result(1) = .ReadLine
        .SkipLine
        .SkipLine
        result(2) = .ReadLine
        .Close
    End With
```

```
    Debug.Print result(1)
    Debug.Print result(2)
End Sub
```

运行上述过程，立即窗口的结果如图 1-51 所示。

以上内容的源代码文件为“实例文档 04.xlsm”。

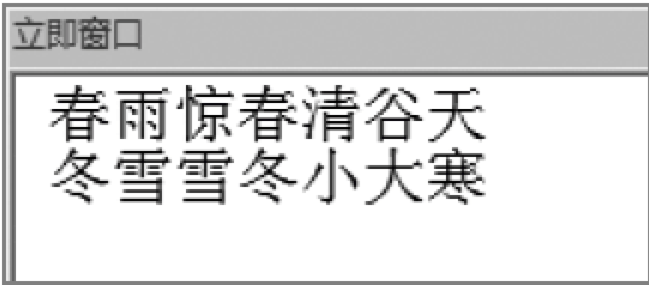


图 1-51　跨行读取内容

1.4　使用 ADODB.Stream 实现文件读写

对于 UTF-8 编码的含中文的文本文件，无论是以传统方式，还是用 FSO 的 TextStream 对象打开，读取到的内容会出现乱码。

本书源文件“ utf-8File.txt”文件中也是一首二十四节气歌，但是该文件的编码格式是 UTF-8，如图 1-52 所示。



图 1-52　UTF-8 编码的文本文件

使用前面讲过的 FSO 方式读取该文件，读取到的是乱码，如图 1-53 所示。

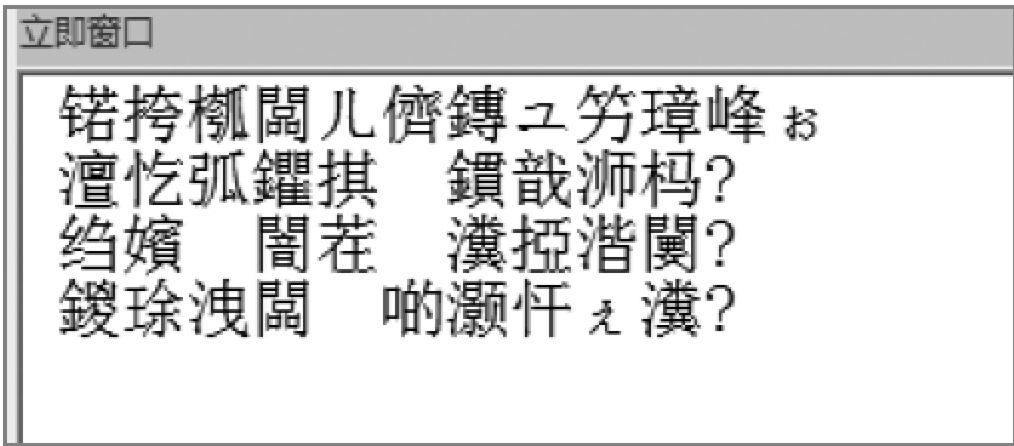


图 1-53　不期望的读取结果

下面讲述一种能够按照指定编码打开文件的方式：ADODB.Stream 对象。

1.4.1 对象的引入

ADODB 是一个非常重要的对象，经常用于数据库操作，在后面的章节会探讨 ADODB 操作数据库方面的知识，本节介绍一下 ADODB.Stream 读写文件。

前期绑定：工程中添加引用“Microsoft ActiveX Data Objects 2.8”，如图 1-54 所示。

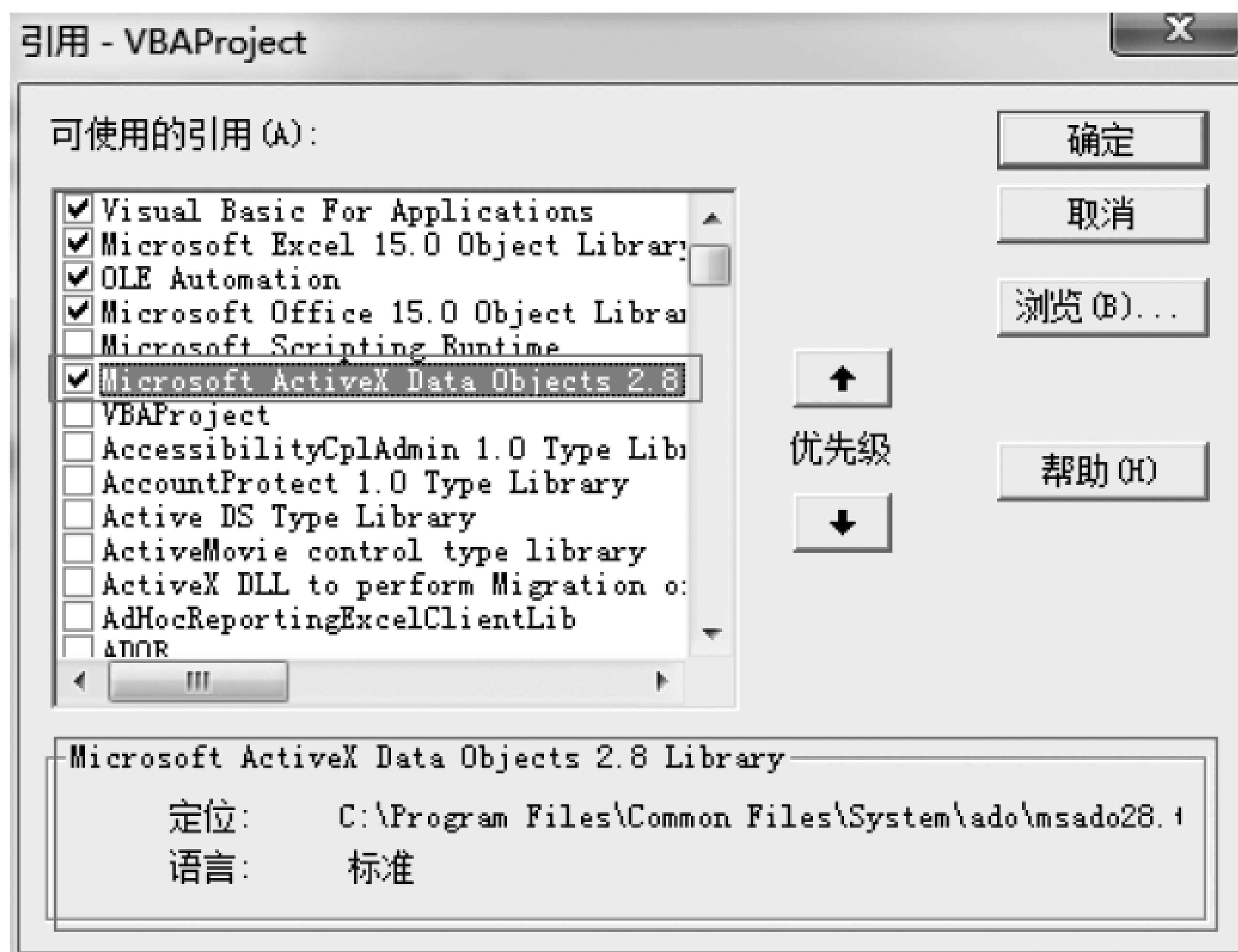


图 1-54 添加外部引用

代码中声明：Dim AStream As New ADODB.Stream，会看到自动成员提示，说明绑定成功，如图 1-55 所示。

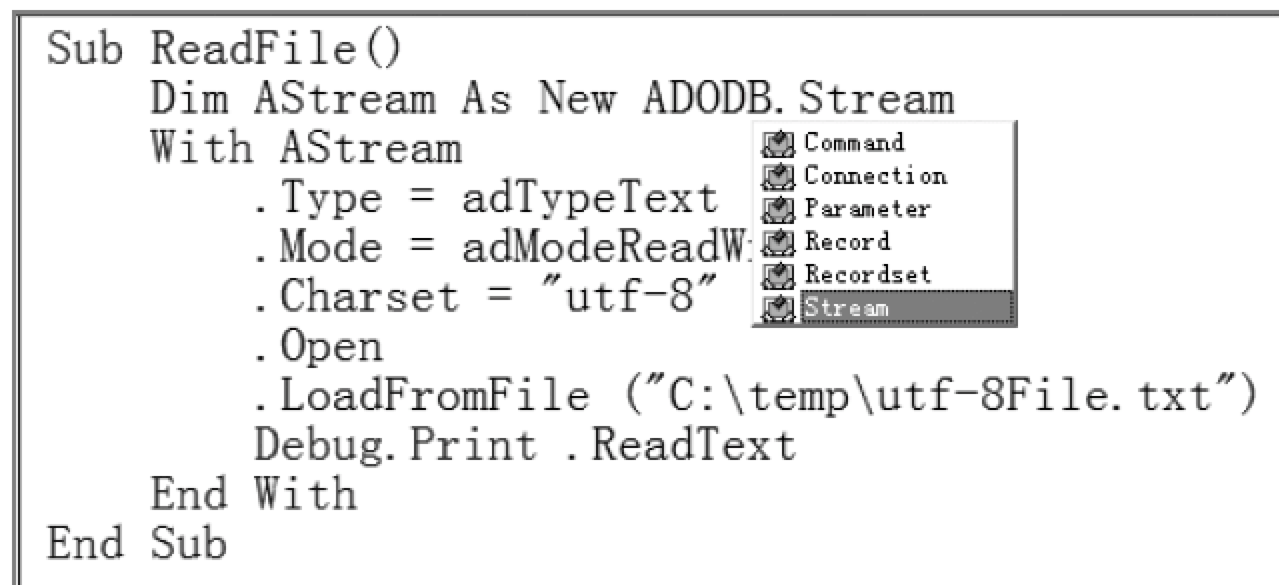


图 1-55 使用 ADODB.Stream

后期绑定：在工程中不添加 ADODB 的前提下，使用 CreateObject("ADODB.Stream") 创建一个新的 Stream 对象。

1.4.2 读取文本文件

ADODB.Stream 对象通过 LoadFromFile 方法载入文本文件，然后用 ReadText 方法读取所有内容。

如果 ReadText 后面不带参数，则相当于 FSO 中的 ReadAll，读取全部内容，如果是 ReadText i，则表示读取 i 个字符。

但是在装载文件之前，必须预设 ADODB.Stream 对象的若干属性。

- Type 属性：读写文本文件用 adTypeText(2)，读写二进制文件用 adTypeBinary(1)。
- Mode 属性：使用 adModeReadWrite(3)，可读写。
- CharSet 属性：指定文件编码，要根据文本文件的编码来设定。

以下过程读取 UTF-8 格式的二十四节气歌的前四个字符。

```
Sub ReadFile()
    Dim AStream As New ADODB.Stream
    Dim result As String
    With AStream
        .Type = adTypeText
        .Mode = adModeReadWrite
        .Charset = "utf-8"
        .Open
        .LoadFromFile ("C:\temp\utf-8File.txt")
        result = .ReadText(4)
        .Close
    End With
    Debug.Print result
End Sub
```

上述过程的打印结果是：“春雨惊春”。如果把 ReadText(4) 改成 ReadText，则读取出所有内容。

对应的后期绑定代码如下所示。

```
Sub LaterBind()
    Dim AStream As Object
    Dim result As String
    Set AStream = CreateObject("ADODB.Stream")
    With AStream
        .Type = 2
        .Mode = 3
        .Charset = "utf-8"
        .Open
        .LoadFromFile ("C:\temp\utf-8File.txt")
        result = .ReadText(4)
        .Close
    End With
    Debug.Print result
End Sub
```

需要注意的是，由于代码中采用了后期绑定方式，并未在工程中添加 ADODB 引用，因此 Type、Mode 等属性不能用枚举常量，只能使用枚举常量的等价值。

1.4.3　写入文本文件

使用 ADODB.Stream 写入文本文件的步骤是：创建对象→设定属性→打开对象→写入内容→保存到文件→关闭对象。

下面的代码把一个字符串重复两次写入文本文件，并保存为 UTF-8 格式。

```
Sub WriteFile()
    Dim AStream As New ADODB.Stream
    Const result As String = "天长地久有没有" & vbNewLine & "浪漫传说说太多"
    With AStream
        .Type = adTypeText
        .Mode = adModeReadWrite
        .Charset = "utf-8"
        .Open
        .WriteText result
        .WriteText result
        .SaveToFile "C:\temp\t.txt", adSaveCreateOverWrite
        .Flush
        .Close
    End With
End Sub
```

代码分析：枚举常量 adSaveCreateOverWrite(2)表示如果目标文件已存在，则覆盖保存。

运行上述过程，再次打开记事本文件，如图1-56 所示。

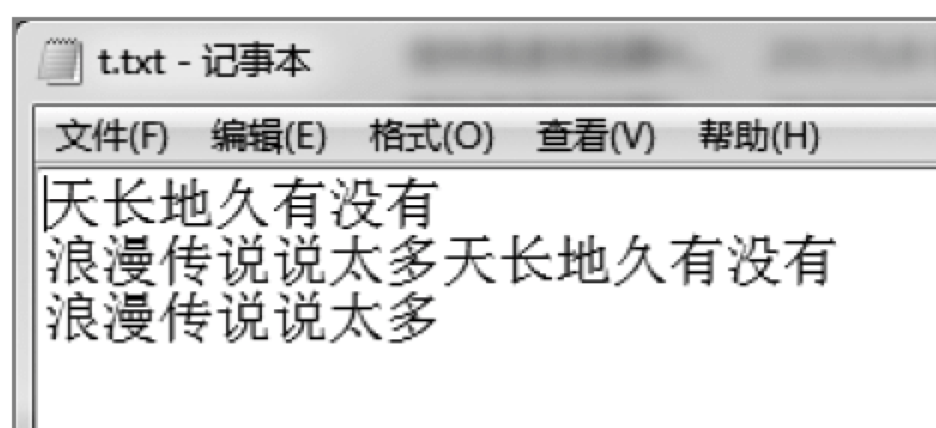


图 1-56 保存为 UTF-8 文件

1.4.4 利用 ADODB.Stream 下载网页附件

除了文本文件以外，ADODB.Stream 还可以读写二进制文件。在二进制文件代码编写方面，需要改动的地方主要有以下两个。

- ❑ Type 属性：需要改为 adTypeBinary。
- ❑ ReadText 方法、WriteText 方法分别更改为 Read、Write。

下面讲解 ADODB.Stream 对象联合使用 XMLHttp 对象，实现网页附件下载到本地的功能。

XMLHttp 对象经常用于向 HTTP 服务器发送请求，其前期绑定方式是向工程中添加引用“Microsoft XML v6.0”，如图 1-57 所示。

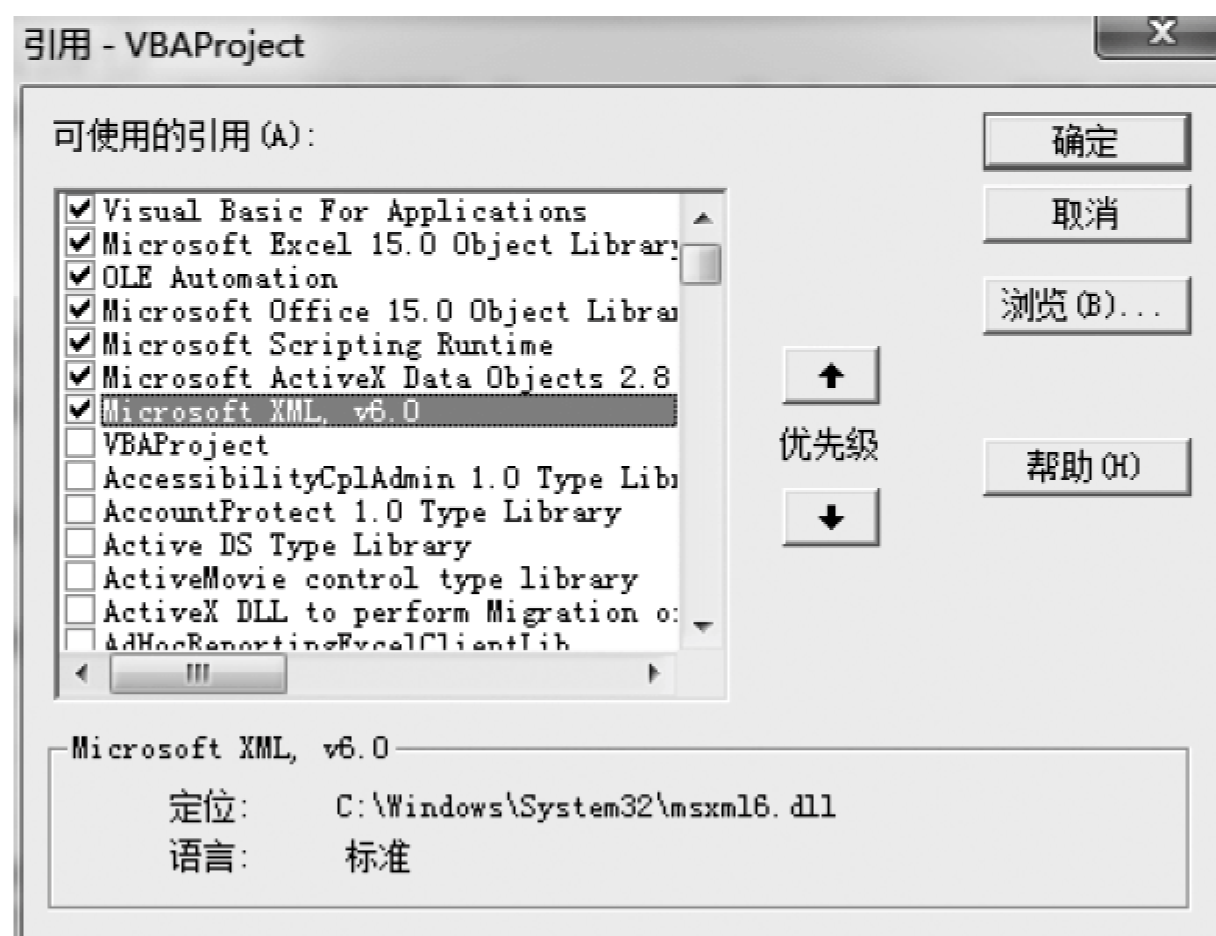


图 1-57 添加外部引用

后期创建对象的方法是：CreateObject("Microsoft.XMLHTTP")。

XMLHttp 对 url 完成请求后，返回的 ResponseBody 是一个未解码的二进制数据，因此，得到这个二进制数据后，用 ADODB.Stream 写入计算机中的文件即可实现附件的下载。

下面的实例从 WinRAR 压缩软件的官方网站下载 WinRAR 5.50 的安装文件。

```
Sub DownLoadFile()
    Dim AStream As New ADODB.Stream, X As New XMLHTTP, Content() As Byte
    With X
        .Open "GET", "http://www.winrar.com.cn/download/wrar550scp.exe", False
        .send
        Do Until .readyState = 4
            DoEvents
        Loop
        Content = .responseBody
    End With
    With AStream
        .Type = adTypeBinary
        .Mode = adModeReadWrite
        .Open
        .Write Content
        .SaveToFile "C:\temp\Winrar550.exe", adSaveCreateOverWrite
        .Close
    End With
End Sub
```

代码分析：Content() 是一个字节数组，用于存储 XMLHttp 返回的 ResponseBody，然后用 ADODB.Stream 的 Write 方法，把 Content 保存为文件。

运行上述过程后，C:\temp\ 文件夹下多了一个 Winrar550.exe 文件。

以上内容的源代码文件为“实例文档 05.xlsm”。

1.5　本章小结

本章讲解了使用传统方式和使用 FSO 处理文件、文件夹的方法以及文本文件内容的读写方法等知识点。

对于读写 UTF-8 格式的文本文件，为了避免读出的内容出现乱码，使用 ADODB. Stream 对象实现了二进制字节数组和字符串的正确转换。

第 2 章

文件系统自动化

编程开发时，经常要对计算机中的文件、文件夹用代码自动打开、自动选中或者自动用默认的浏览器打开某个网页。在 VBA 中可以使用 Shell 函数来代替这些手工操作。

如果要实现自动发送按键、自动创建桌面快捷方式、自动读写注册表，使用 WshShell 对象。

因此，本章将介绍 Shell 函数实现文件系统自动化、用代码操作注册表、自动发送按键三大部分。

本章用到的外部引用和重要对象如下。

- Windows Script Host Object Model
  - IWshRuntimeLibrary.WshShell
  - IWshRuntimeLibrary.WshNetwork

2.1 Shell 函数

Shell 用于执行一个可执行文件，返回一个 Variant (Double) 类型的结果，如果成功，代表这个程序的任务 ID，若不成功，则会返回 0。

Shell 的语法格式是：

```
Shell(PathName,WindowStyle)
```

具体的参数说明：

- PathName：是一个字符串，必须包含一个可执行文件的名称，后面还可以加上一些命令参数，各个参数之间用空格隔开。
- WindowStyle：枚举常量，用来规定窗口样式，如表 2-1 所示。

表 2-1　Shell 命令的窗口样式常量

常　　量	描　　述
vbHide	窗口被隐藏，且焦点会移到隐式窗口
vbnormalFocus	窗口具有焦点，且会还原到它原来的大小和位置

续表

常　　量	描　　述
vbMinimizedFocus	窗口会以一个具有焦点的图标来显示
vbMaximizedFocus	窗口是一个具有焦点的最大化窗口
vbNormalNoFocus	窗口会被还原到最近使用的大小和位置，而当前活动的窗口仍然保持活动
vbMinimizedNoFocus	窗口会以一个图标来显示，而当前活动的窗口仍然保持活动

需要注意的是，在 VBA 或 VB6 编程中，Shell 函数是内置函数，不需要添加任何外部引用，也不需要创建对象。

Shell 函数的功能可以这样理解，就是代替手工去打开某个应用程序或文档。

实际上，屏幕上面的窗口的根源绝大多数都来源于一个可执行文件（扩展名为 .exe）。例如微软的 Word，其窗口根本路径是 Office 安装路径下的 WINWORD.EXE 文件，当打开一个文本文件时，其根源文件是 C:\Windows\System32\notepad.exe。那么 Windows 系统中的可执行文件，一部分是系统自带的，例如记事本、计算器、注册表编辑器等，它们都处于 System32 文件夹下，是系统内置可执行文件。而另一部分如 QQ、Excel 这些应用软件则是后期安装上的，根源文件往往位于用户指定的路径。

为了更好地理解 Shell 函数的原理，下面使用计算机的“运行”窗口完成一些动作。按下快捷键【 Windows + R 】，在屏幕左下角弹出“运行”窗口，如图 2-1 所示。

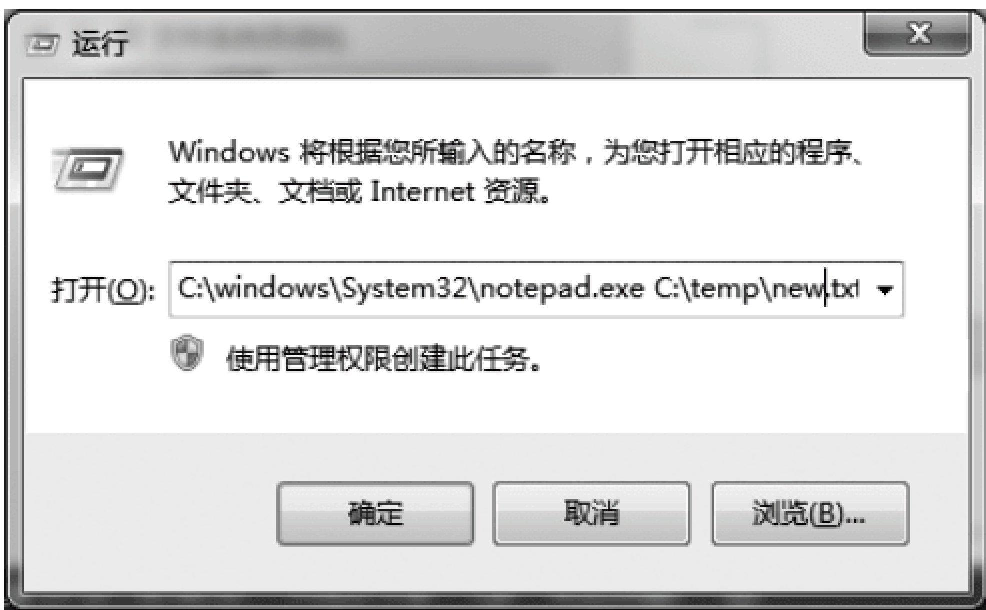


图 2-1　运行窗口

在路径框中输入：C:\windows\System32\notepad.exe C:\temp\new.txt，单击“确定”按钮，就可以看到桌面自动弹出记事本，并能看到其中的内容。

我们仔细分析路径的构成，C:\windows\System32\notepad.exe 其实就是系统文件记事本的根源文件，空格后面的路径是命令参数部分，也就是告诉记事本程序，要打开这个文件，而不是其他的文本文件。

针对以上实例，可以使用下面的 Shell 函数来达到同样目的。只需要把“运行”中的内容传递给 Shell 函数即可。

```
Sub Test1()
    Shell "C:\windows\System32\notepad.exe C:\temp\new.txt", vbMaximizedFocus
End Sub
```

上述代码中的 vbMaximizedFocus 表示打开的记事本窗口取得焦点，并且最大化记

事本窗口，如果改为 vbHide，则打开的新窗口在屏幕上看不见，在某些场合下这样是很危险的。

对于上述过程，我们要认识到以下几点。

- ❑ Shell 函数的参数由可执行文件的路径以及命令参数路径构成。
- ❑ 可执行文件的路径，根据需要可以尽可能缩短简化。
- ❑ 命令参数不是必需的，如果不带命令参数，则默认只打开该应用程序。

由于上例中的 notepad.exe 处于系统文件夹中，系统的环境变量中一定有 C:\windows\System32 这个路径。因此，Shell 函数还可以省略所在路径，以及后面的扩展名，从而简化为：Shell "notepad C:\temp\new.txt", vbMaximizedFocus。

如果不打开具体的文件，只打开记事本，则不需带命令参数。进一步简化为：

```
Shell "notepad"
```

根据以上心得，我们尝试用下面的代码打开一个 Word 文档。

```
Shell "WinWord C:\temp\ 公务员 .docx"
```

代码中的 WinWord 是 Office 安装文件夹中 Word 程序的主文件，如图 2-2 所示。

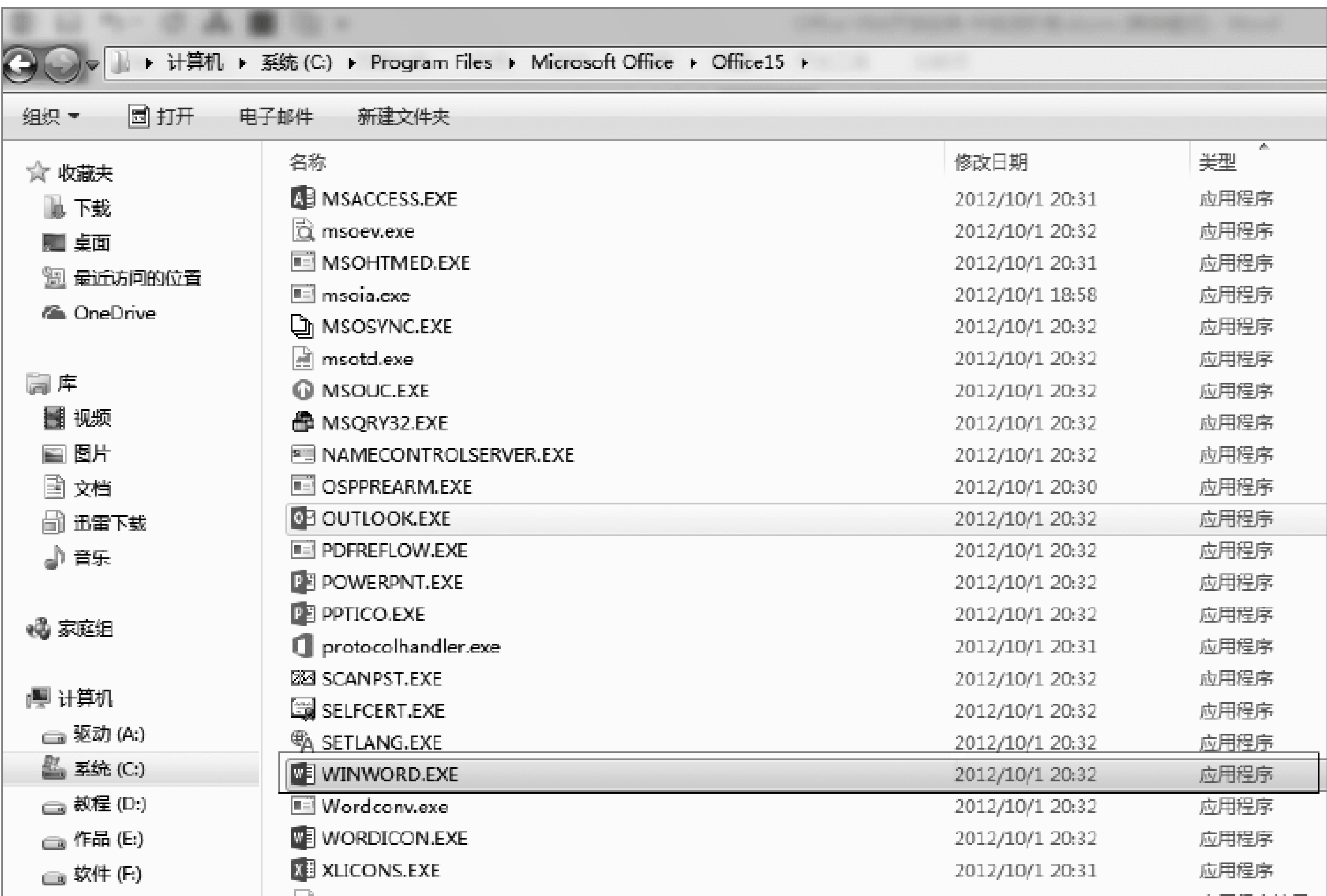


图 2-2　Word 的执行文件

2.1.1　System32 中常用的可执行文件

Windows 系统中，很多实用的工具都处于 System32 这个文件夹下，因此可以利用 Shell 函数调用这些可执行文件，完成一些特定的任务。比较常用的可执行文件如表 2-2 所示。

表 2-2　常用的可执行文件

文　件　名	中　文　名	功 能 描 述
calc.exe	计算器	—
cmd.exe	DOS 窗口	执行 DOS 命令

续表

文　件　名	中　文　名	功 能 描 述
control.exe	控制面板	设置系统性能、卸载程序等
explorer.exe[1]	我的电脑	资源管理器
msconfig.exe	系统实用配置程序	配置开机启动项等
mspaint.exe	画图	图片编辑
notepad.exe	记事本	文本文件读写
regedt32.exe	注册表编辑器	—
regsvr32.exe	注册	注册和取消注册文件
shutdown.exe	关机	自动关机
taskmgr.exe	任务管理器	结束应用程序等
taskkill.exe	结束任务	—

① 这类文件有可能不在 System32 文件夹中。

2.1.2　执行 DOS 命令

DOS 命令的可执行文件是 cmd.exe，用 Shell 函数调用 DOS 命令后，会弹出黑屏窗口，当 DOS 命令执行完毕后，有可能自动退出黑屏，也有可能停留，这取决于 cmd 的参数。/c 表示自动退出，/k 表示停留。

下面的代码，首先切换磁盘分区到 D 盘，其次切换当前目录到 Download 文件夹，最后用 Shell 函数调用 DOS 窗口，用 dir 命令列举出该文件夹中的内容。

```
Sub 调用 DOS()
    ChDrive "D:"
    ChDir "D:\Download"
    Shell "cmd /k dir", vbNormalFocus
End Sub
```

由于 cmd 的参数使用了 /k，因此，当 dir 命令执行完后，黑屏仍然停留在屏幕上，如图 2-3 所示。

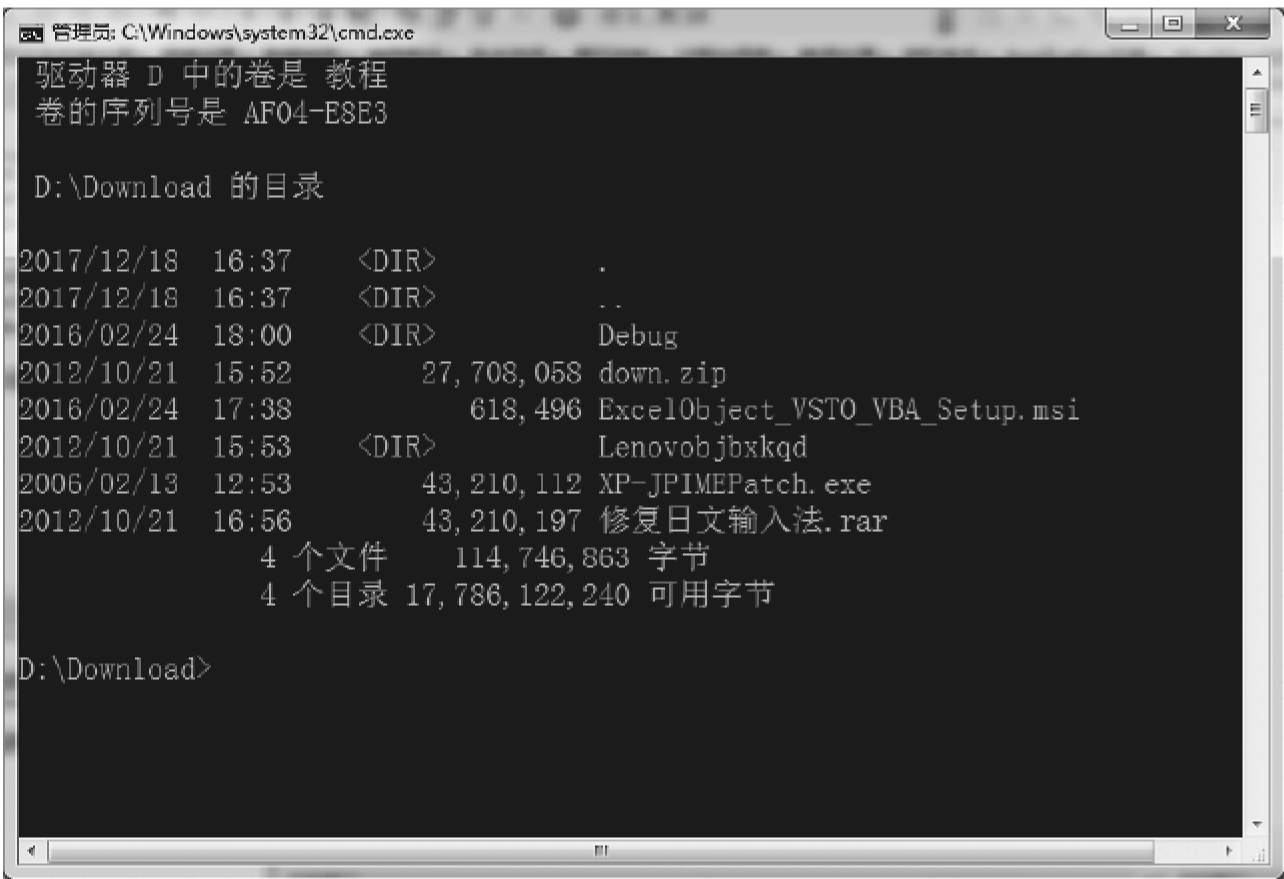


图 2-3　Shell 调用 DOS 命令

DOS 命令众多，此处仅举几个典型的例子。

下面的代码用 DOS 命令自动把 D: 盘根目录下的 VMware10.7z 压缩包（该文件大约 470MB）复制到 D:\Download 文件夹中，并试图把复制后的文件重命名。

```
Sub 复制文件()
    Shell "cmd /k copy D:\VMware10.7z D:\Download\", vbNormalFocus
    Name "D:\Download\VMware10.7z" As "D:\Download\VMwareNew.7z"
End Sub
```

运行上述过程，当运行到第二行时，弹出如下的运行时错误，如图 2-4 所示。

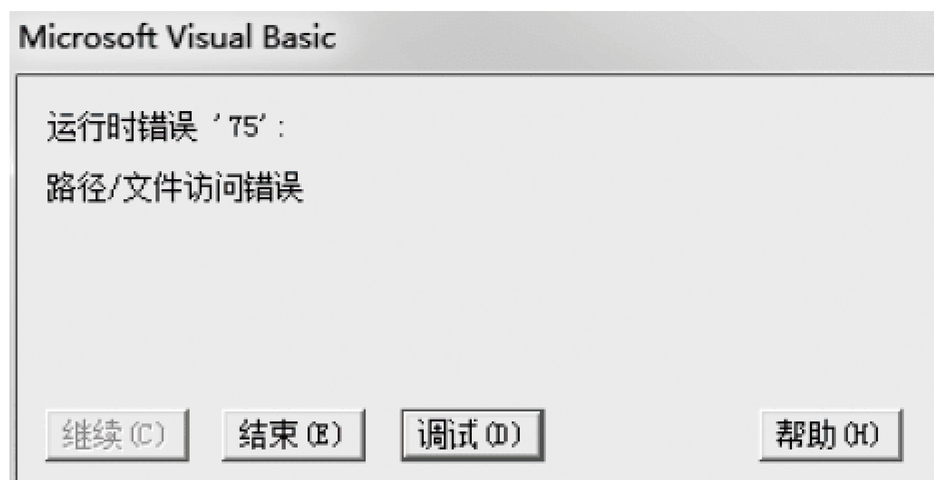


图 2-4　不可移动或重命名

结束程序后，可以看到 Download 文件夹下确实多了一个文件。也就是说，复制操作成功地执行了，但是重命名失败。

导致操作失败的原因是，Shell 函数是异步执行的，也就是说，复制文件的宿主程序是 cmd 文件，并不是 VBA 代码。因此 VBA 并不能监测到 cmd 那边复制操作是否已经完成。换句话说，在 VBA 中执行一句 Shell 语句几乎不花时间，而实际上的复制动作可能持续好几秒或好几分钟。由于 VBA 监测不到复制的状态，就立即运行后续的代码，进行重命名，出现运行时错误也就可以理解了。

2.1.3　认识 Shell 函数的异步

为了提高 Shell 函数的健壮性，利用 Shell 函数的返回值，再配合一些 API 函数，可以让程序在执行 Shell 函数的时候智能等待。也就是说，只有 Shell 函数中的动作已完成时，才往下继续执行代码，否则在空循环中等待。

在标准模块中写入如下代码。

```
    Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess As
Long, lpExitCode As Long) As Long
    Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
    Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As
Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Const PROCESS_QUERY_INFORMATION = &H400
    Const STILL_ALIVE = &H103

    Sub CopyFile()
        Dim Pid As Long, ExitCode As Long
        Pid = Shell("cmd.exe /c copy D:\VMware10.7z C:\lib\", vbNormalFocus)
        hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, 0, Pid)
        Debug.Print "复制进行中...."
        Do
            Call GetExitCodeProcess(hProcess, ExitCode)
            DoEvents
        Loop While ExitCode = STILL_ALIVE
        Call CloseHandle(hProcess)
```

```
    Debug.Print "复制已完成！"
    Name "C:\lib\VMware10.7z" As "C:\lib\VMware10_刘永富.7z"
End Sub
```

代码分析：当文件正在复制时，代码中的 ExitCode 和常量 STILL_ALIVE 是相等的，都是非零值，此时跳不出 Do 循环体。

当复制动作完成时，ExitCode 变为 0，Do 循环的条件不再成立，就跳出并执行后续重命名的代码。

运行上述 CopyFile 过程，即可进行先复制文件，然后重命名复制的文件，如图 2-5 所示。

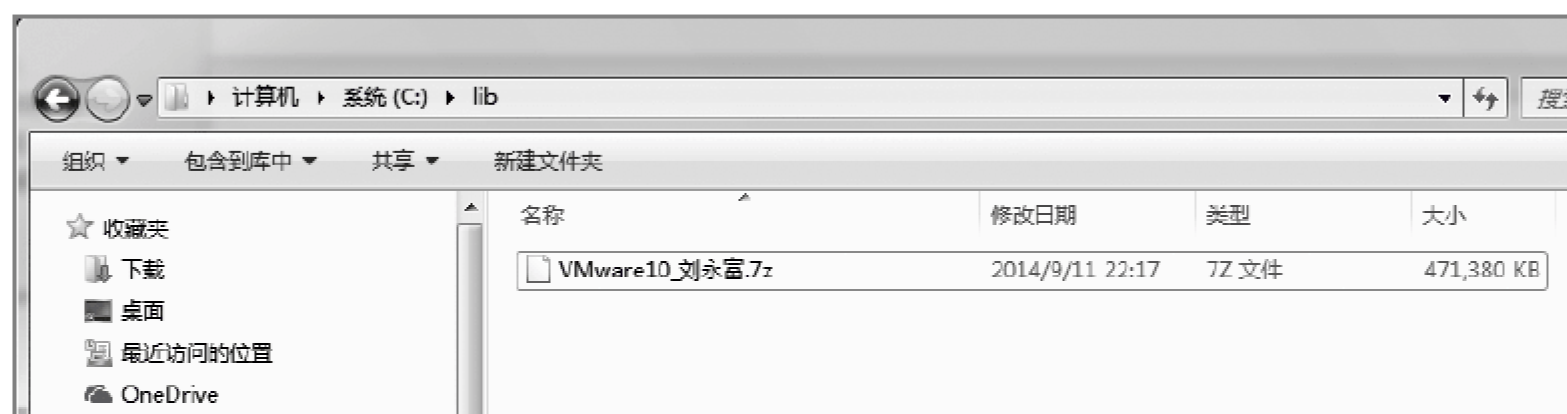


图 2-5　处理 Shell 的异步

因此，只要 Shell 函数中的新窗口不关闭，就不会跳出 Do 循环，读者可以把上述代码中的 Pid = Shell("cmd.exe /c copy D:\VMware10.7z C:\lib\", vbNormalFocus)，更改为 Pid = Shell("notepad.exe", vbNormalFocus)，再试一次，可以发现，弹出的记事本窗口如果不手工关闭，程序就阻滞在 Do 循环体中。

2.1.4　处理 Shell 函数中的空格

计算机的文件夹或文件名中经常包含空格，如果把包含空格的路径作为参数传递给 Shell 函数，该函数会以空格作为分隔符，把一个路径理解为多个参数的连接。

例如，试图用 Word 2013 打开 C:\temp 文件夹下的“公 务 员 .docx”，如果写成如下形式。

```
Shell "C:\Program Files\Microsoft Office\Office15\WINWORD.EXE C:\temp\公 务 员.docx", vbNormalFocus
```

执行时，并不能打开名称或路径带空格的文件，而是弹出如图 2-6 所示错误对话框。

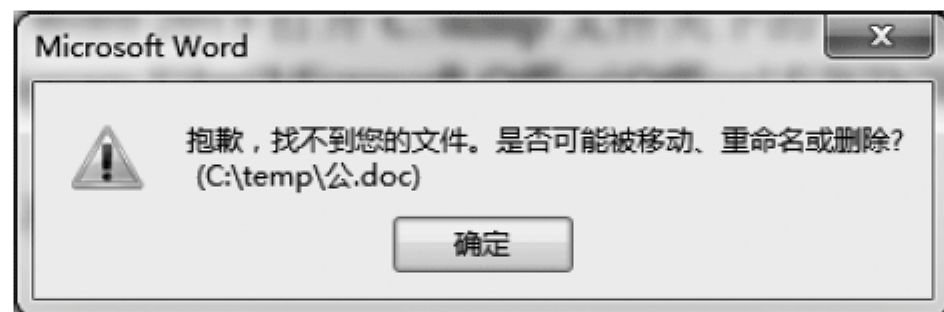


图 2-6　文件或路径中包含空格

这是因为不仅 Word 的路径包含空格，而且计划打开的文档名称也有空格，Shell 函数无法解释这些参数的含义，因此不能打开。

如果为 Shell 函数中各路径事先用双引号包起来，就不会有上述麻烦。

自定义函数 AddQuote 的作用是给路径两侧加上双引号，并且添补一个空格，以防止连在一起。

```
Private Function AddQuote(path As String) As String
    AddQuote = Chr(34) & path & Chr(34) & " "
End Function
```

```
Sub 处理空格 ()
    Dim exepath As String, param As String
    exepath = "C:\Program Files\Microsoft Office\Office15\WINWORD.EXE"
    param = "C:\temp\ 公 务 员 .docx"
    exepath = AddQuote(exepath)
    param = AddQuote(param)
    Shell exepath & param, vbNormalFocus
End Sub
```

运行上面的“处理空格”过程，该文档可以正常地在 Word 中打开。

2.1.5 自动打开控制面板

控制面板的主文件是位于 System32 文件夹下的 control.exe，因此只需要执行：

```
Shell "control.exe", vbNormalFocus
```

就可以打开控制面板的主页，如图 2-7 所示。



图 2-7 自动打开控制面板

如果要直接打开控制面板中特定的一项，需要添加命令参数。

```
Shell "control.exe appwiz.cpl", vbNormalFocus
```

上述代码表示直接打开控制面板中的“卸载程序”，命令参数 appwiz.cpl 是 Application Wizard 的缩写，意思是程序向导，如图 2-8 所示。

因此，只需要把 Shell 函数命令参数中的 cpl 文件更改一下，就可以打开控制面板中的其他各项。常用的有如下这些。

卸载程序：Shell "Control.exe " & "appwiz.cpl", vbMaximizedFocus

显示属性：Shell "Control.exe " & "desk.cpl", vbMaximizedFocus

浏览器属性：Shell "Control.exe " & "inetcpl.cpl", vbMaximizedFocus



图 2-8　控制面板中的项目

区域和语言：Shell "Control.exe " & "intl.cpl", vbMaximizedFocus

声音和音频：Shell "Control.exe " & "mmsys.cpl", vbMaximizedFocus

网络连接：Shell "Control.exe " & "ncpa.cpl", vbMaximizedFocus

用户账户：Shell "Control.exe " & "nusrmgr.cpl", vbMaximizedFocus

电源选项：Shell "Control.exe " & "powercfg.cpl", vbMaximizedFocus

计算机属性：Shell "Control.exe " & "sysdm.cpl", vbMaximizedFocus

日期和时间：Shell "Control.exe " & "timedate.cpl", vbMaximizedFocus

安全中心：Shell "Control.exe " & "wscui.cpl", vbMaximizedFocus

2.1.6　打开资源管理器

资源管理器是以树状结构显示计算机中的文件系统的窗口，在 Windows XP 系统中双击“我的电脑”，在 Windows 7 系统中双击“计算机”，或者按下快捷键【 Windows + E 】，都可以打开资源管理器。

资源管理器的主文件为 explorer.exe。

在实际编程中，经常用到自动打开某个文件夹，或者自动选中文件、文件夹的操作。如下面的代码。

```
Shell "explorer.exe C:\temp\成绩表.pdf", vbNormalFocus
```

上述代码表示用系统默认程序打开指定的 PDF 文件，这相当于用鼠标双击了计算机中的“成绩表.pdf”文件，如果计算机中安装的是 Adobe Acrobat，那么用该软件打开 PDF 文件。对于其他扩展名的文件也是如此，因此这个代码实用价值很高。

```
Shell "explorer.exe C:\temp\", vbNormalFocus
```

上述代码表示打开 temp 文件夹，相当于用鼠标双击了该文件夹，进入该文件夹内部。

针对以上两句代码，如果添加参数“/select,”，那么不是打开文件或文件夹，而是选中。

```
Shell "explorer.exe /select, C:\temp\成绩表.pdf", vbNormalFocus
```

上述代码表示显示资源管理器，并自动选中文件，如图 2-9 所示。

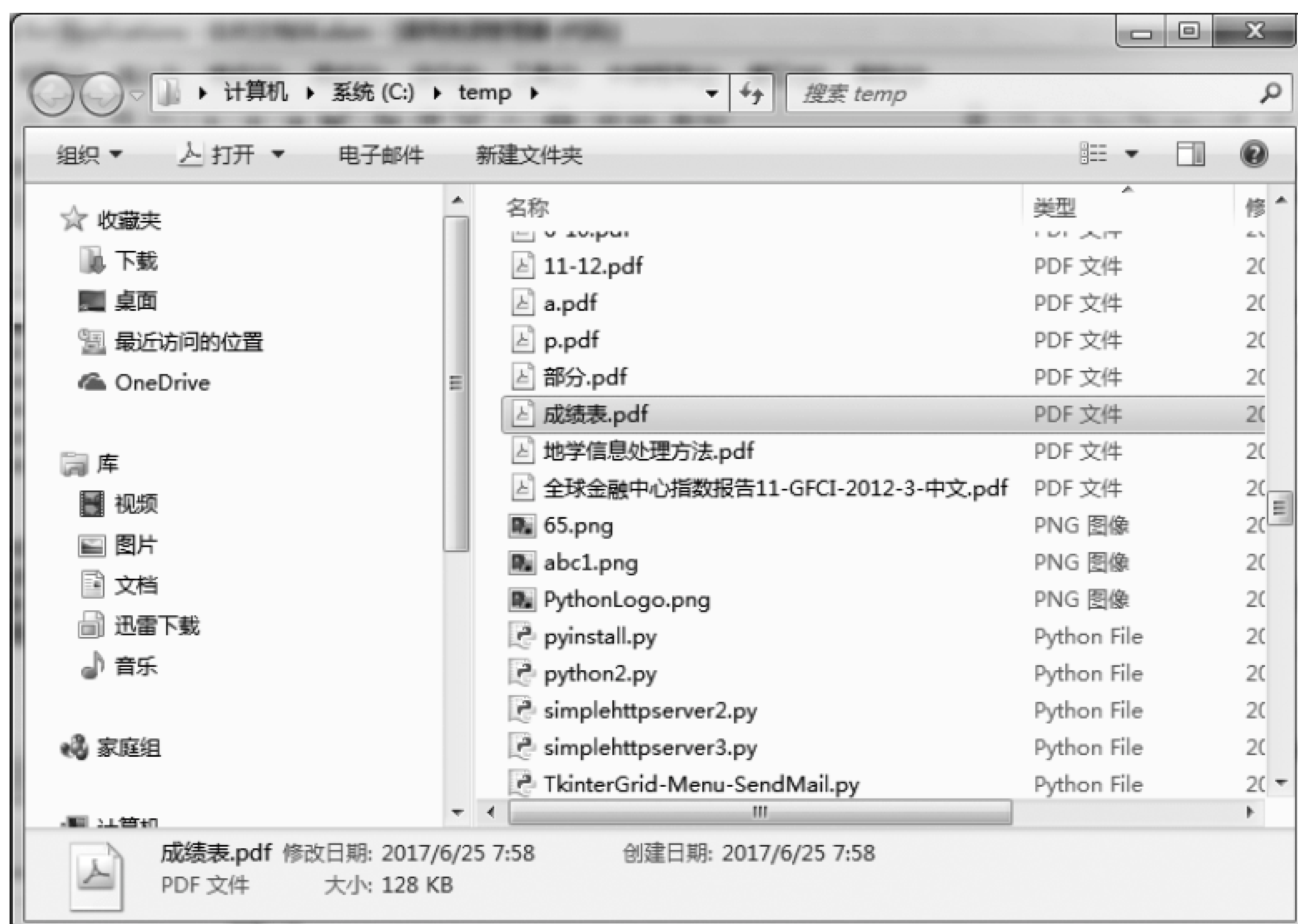


图 2-9 在资源管理器中自动选中文件

```
Shell "explorer.exe /select, C:\temp\", vbNormalFocus
```

上述代码表示自动选中 temp 文件夹，而不打开。

此外，利用 explorer 还可以自动打开指定 url 的网页。

```
Shell "explorer.exe http://vba.mahoupao.net/forum.php", vbNormalFocus
```

上述代码会用计算机默认的网页浏览器打开一个论坛。

2.1.7 注册 ocx 文件和 dll 文件

ocx 文件是指对象类别扩充组件。计算机中扩展名为 .ocx 的文件，不能直接双击执行，一般要把 ocx 控件插入到窗体中使用。例如在 VBA 的 UserForm 中，可以插入 CommonDialog、DataGrid 这些 ActiveX 控件，如图 2-10 所示。

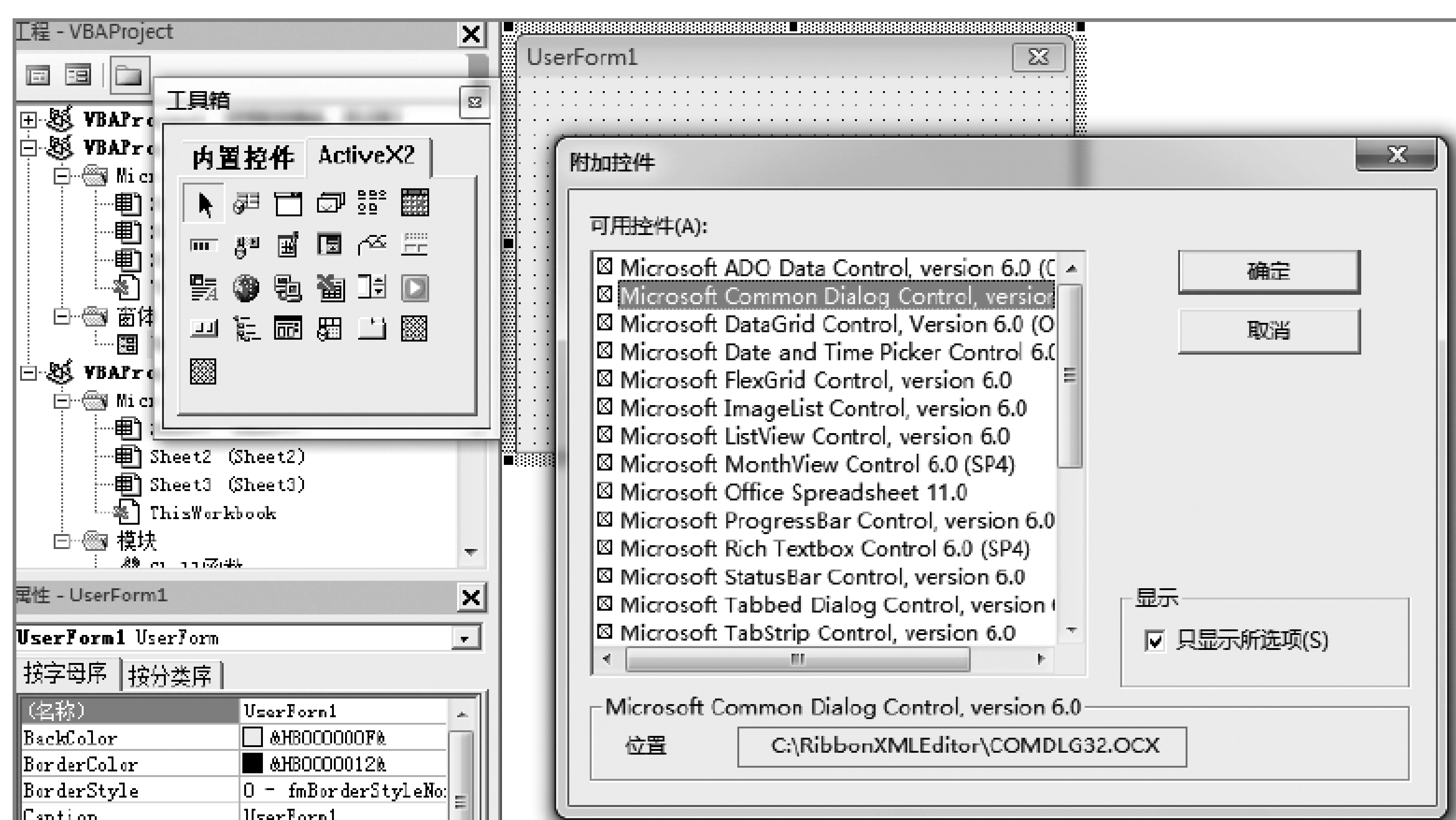


图 2-10 用户窗体中使用 ocx 控件

这些 ocx 文件大多数是微软公司开发的性能优良的控件，插入到窗体后，可以让编程更加简单，界面更加专业、美观。

此外，用户根据需要也可以自己制作专用的 ocx 控件。但是，ocx 的移植，也就是说从一台计算机把 ocx 控件复制到另一台计算机，是不能直接使用的，必须在目标计算机上注册。因为未注册的 ocx 控件不会显示在“附加控件”对话框中，也就无法装入窗体。所谓注册，就是为该控件分配一个 GUID，并保存于注册表中。反注册，就是从注册表中移除该控件的有关信息，使其无法在编程环境中访问。

注册和反注册文件，可以用 Shell 函数调用 System32 文件夹下的 regsvr32.exe 文件实现。

注册 ocx 文件的语法格式如下。

```
Shell "regsvr32.exe ocx 文件的路径 "
```

下面尝试注册一款笔者开发的 TreeviewExplorer.ocx 控件，并在 VBA 中运行。

```
Shell "regsvr32.exe E:\TreeviewExplorer\TreeviewExplorer.ocx"
```

执行上述过程会弹出注册成功的提示框，如图 2-11 所示。

如果要屏蔽注册成功的对话框，可以在 regsvr32.exe 后面添加 /s 参数，也就是改为如下形式。

```
Shell "regsvr32.exe /s E:\TreeviewExplorer\
TreeviewExplorer.ocx"
```

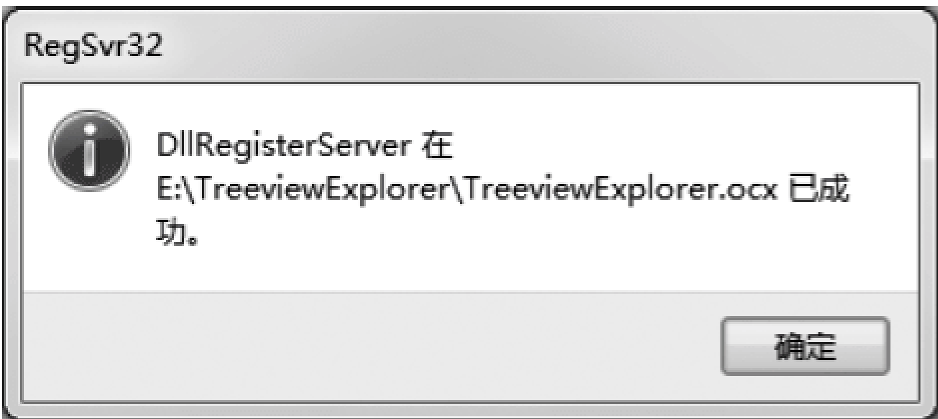


图 2-11　成功注册 ocx 控件

那么，注册成功后，到底引起了哪些变化呢？

运　行 Shell "regedt32.exe" ,vbNormalFocus，自动打开注册表编辑器，在查找对话框中输入关键字，可以快速找到该控件的注册信息，如图 2-12 所示。

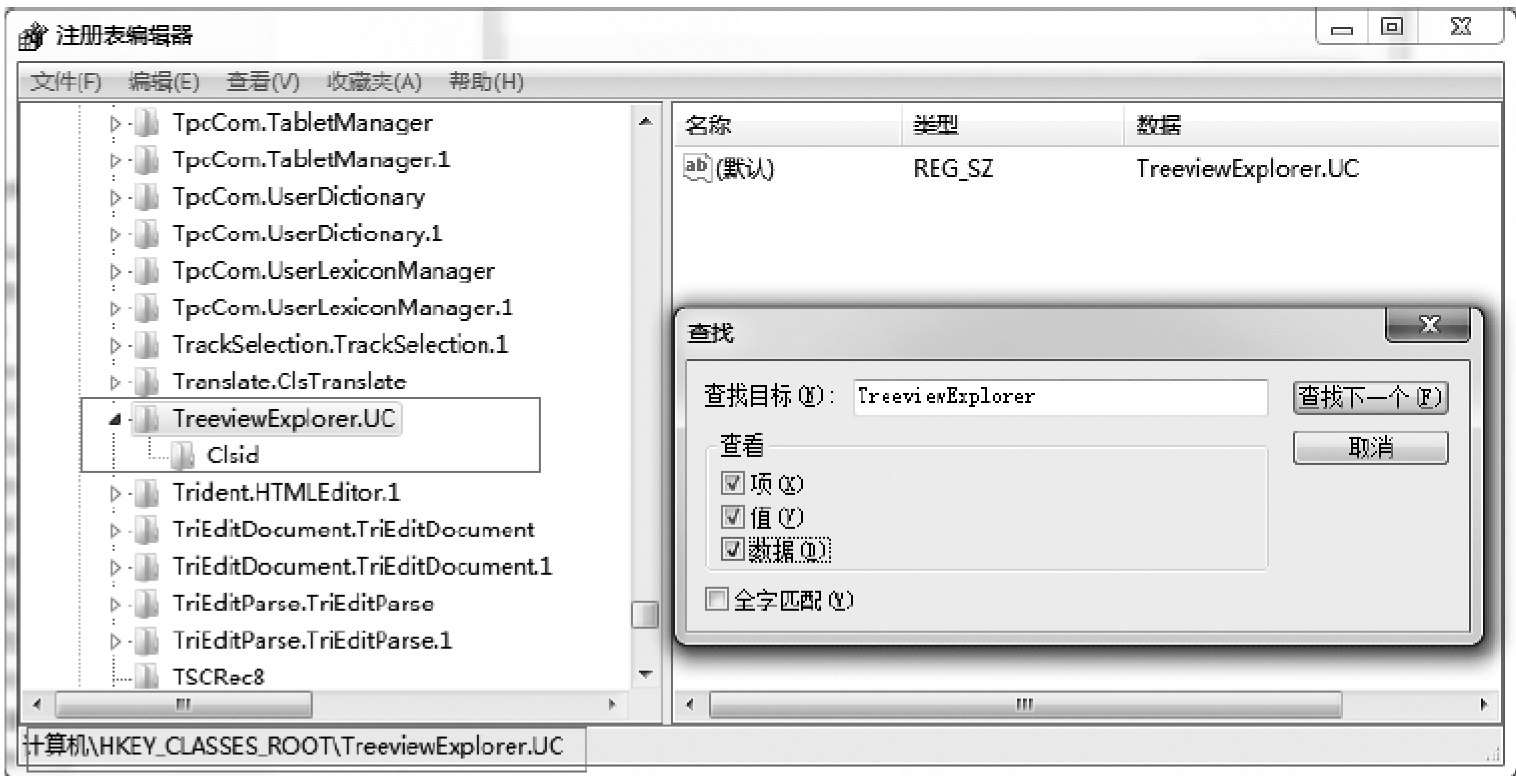


图 2-12　ocx 控件的信息写入注册表

可以看出注册信息位于：HKEY_CLASSES_ROOT\TreeviewExplorer.UC。

注册成功的控件，更重要的变化在于能够在各种窗体中使用该控件。下面是 VBA 的用户窗体中的“附加控件”对话框，在该项前面勾选，即可把自定义控件添加到“控件工具箱”中，从而可以插入窗体中，如图 2-13 所示。

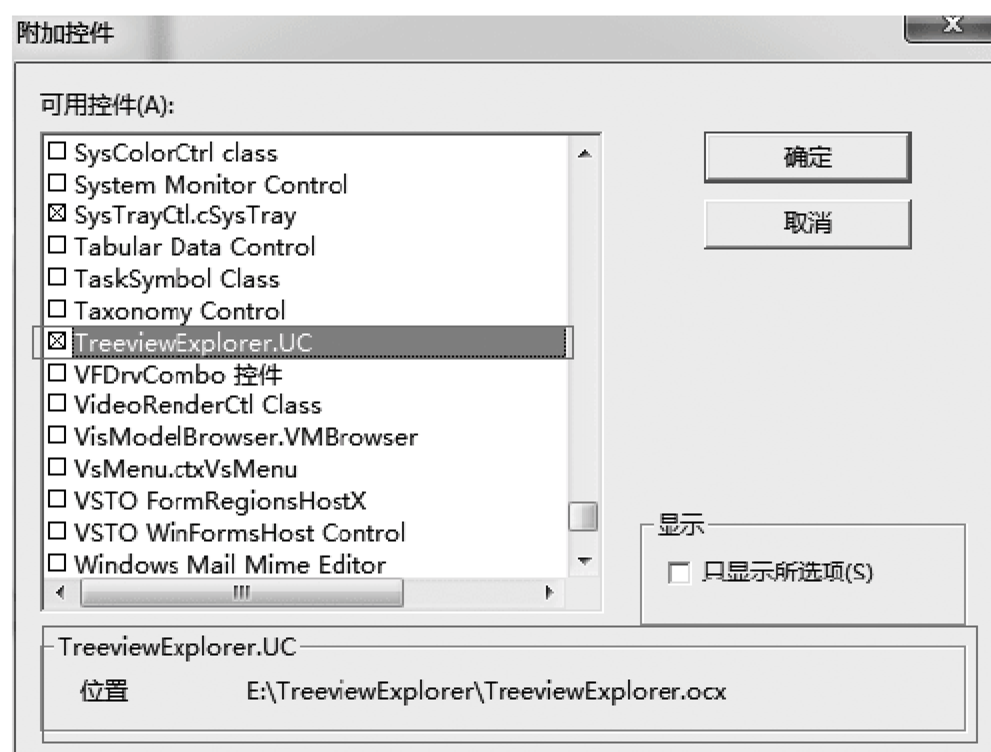


图 2-13 使用自定义控件

控件的反注册也很简单，只需要在上述注册的代码中插入一个 /u 参数，就是反注册。

```
Shell "regsvr32.exe /u E:\TreeviewExplorer\TreeviewExplorer.ocx"
```

如果不弹出提示对话框，更改为如下代码。

```
Shell "regsvr32.exe /s /u E:\TreeviewExplorer\TreeviewExplorer.ocx"
```

控件被反注册以后，注册表中会删除该控件的信息，在“附加控件”对话框中也找不到该控件。

dll 文件，即动态链接库文件（Dynamic Link Library），也可以称为类库，与 ocx 文件一样，不能直接运行。dll 文件中主要封装了一些函数和代码。Office 的 COM 加载项就是一种扩展名为 .dll 的文件。

在 VBA 编程中，可以向 VBA 工程的引用中添加 dll 文件，从而使用 dll 文件中的功能和函数。

dll 文件的注册、反注册方法和 ocx 的代码是一模一样的，只需要把 ocx 控件的路径更改为 dll 文件的路径即可。

OfficeDll 是笔者开发的一款功能丰富的动态链接库，把该文件复制到目标计算机后，运行下面的代码进行注册。

```
Shell "regsvr32.exe E:\OfficeDll\OfficeDll.dll"
```

注册成功后，单击 VBA 编辑器的菜单【工具 / 引用】，在“可使用的引用”列表中可以看到该动态链接库，如图 2-14 所示。

添加引用成功后，编写代码，可以看到该类库的成员列表，如图 2-15 所示。

注意，如果同一台计算机中存在多个同名 ocx 或 dll 文件，以最近注册的为准。

注册和反注册的方法总结如下。

Shell "regsvr32.exe 文件 " 表示注册一个文件，并弹出提示对话框。

Shell "regsvr32.exe /s 文件 " 表示注册一个文件，不弹出提示对话框。

Shell "regsvr32.exe /u 文件 " 表示取消注册一个文件，并弹出提示对话框。

Shell "regsvr32.exe /s /u 文件 " 表示取消注册一个文件，不弹出提示对话框。

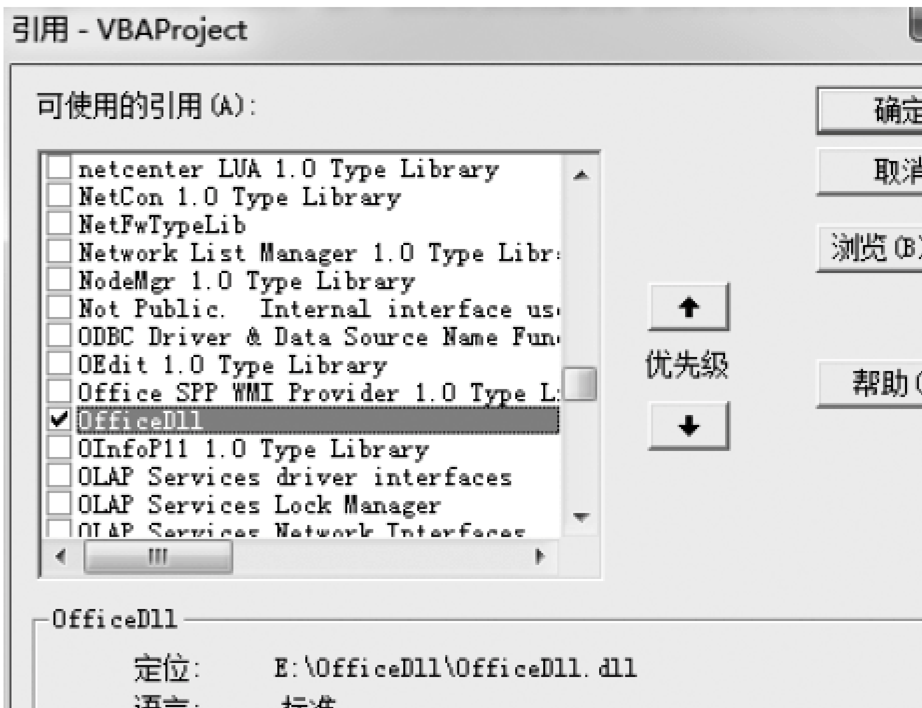


图 2-14　动态链接库的注册和使用

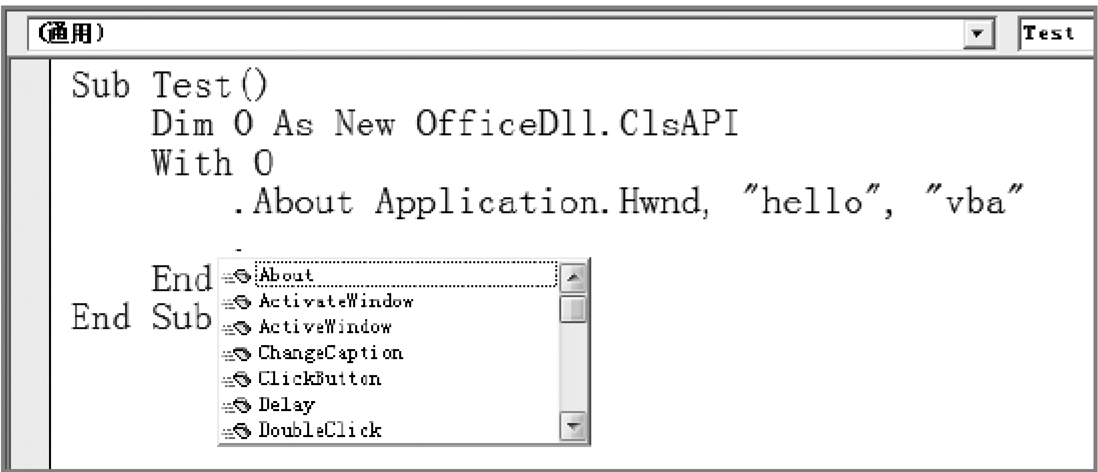


图 2-15　VBA 中使用动态链接库

关于 ocx 控件和动态链接库的开发和应用，本书暂不讨论。

2.1.8　结束进程

在 Windows 系统中按下快捷键【Ctrl+Shift+Esc】可以弹出 Windows 任务管理器，通过 Windows 任务管理器可以强行结束一个进程，如图 2-16 所示。



图 2-16　Windows 任务管理器

假定现在屏幕上打开了计算器，进程列表中一定有 calc.exe 进程，然后在 VBA 中运行如下代码。

```
Shell "taskkill /f /im calc.exe", vbHide
```

可以看到计算器自动被终止。taskkill.exe 也是 System32 文件夹下的一个可执行文件，专门用来终止进程。

2.1.9　自动关机

shutdown.exe 是一个用于关机的系统文件，用 Shell 函数调用 shutdown 可以实现自动关

机、取消关机、自动重启等操作。

```
Sub Test1()
    Shell "shutdown -s", vbNormalFocus          '一分钟后关机
End Sub
Sub Test2()
    Shell "shutdown -r", vbNormalFocus          '一分钟后重启
End Sub
Sub Test3()
    Shell "shutdown -a", vbNormalFocus          '取消计划
End Sub
```

运行上述 Test1 或 Test2，会弹出提示框，如图 2-17 所示。

如果又不想关机或重启，那么需要运行 Test3 取消计划。计划被取消时，屏幕右下角弹出提示，如图 2-18 所示。

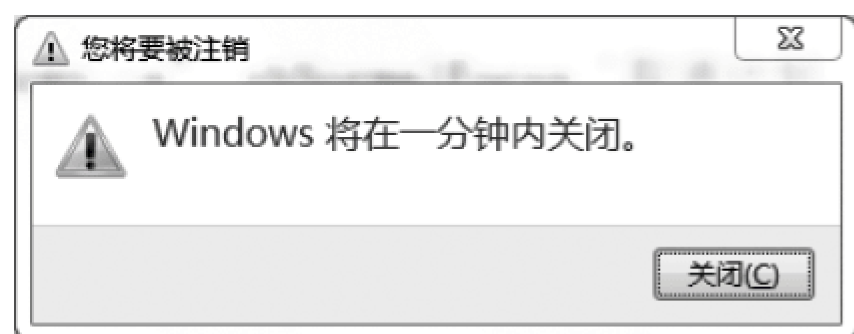


图 2-17　计划关机

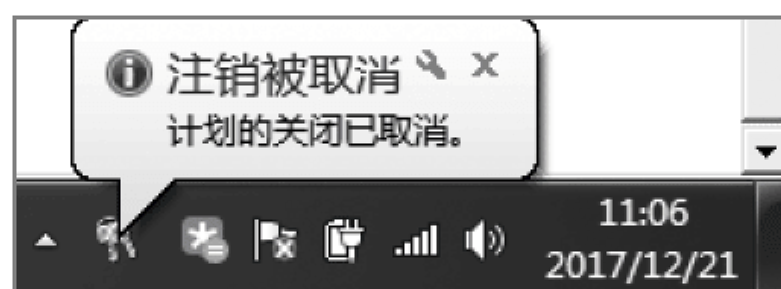


图 2-18　取消计划

在日常办公中，一台计算机经常会连续工作好几天，此时，可以用 Excel VBA 设置一个计划，在未来某一天的某一时刻定时关机。

Application 对象的 OnTime 方法可以在某一时刻准时执行某个过程，因此，运行下面的 MySchedule 过程，计算机不会发生任何变化，但是到了指定的时刻，会准时调用 AutoShutdown 过程，从而实现自动关机。

```
Sub AutoShutdown()
    Shell "shutdown -s", vbNormalFocus          '一分钟后关机
End Sub
Sub MySchedule()
    Application.OnTime #12/21/2017 11:11:00 AM#, "AutoShutdown"
End Sub
```

注意　运行 MySchedule 过程后，Excel 要一直保持打开状态，如果退出 Excel，计划无效。

以上内容的源代码文件为“实例文档 06.xlsm”。

2.2　内置注册表函数

注册表是 Windows 系统中的一个系统数据库，主要存储硬件配置、软件设置等信息。注册表的操作主要有创建注册表项、删除注册表项、修改注册表项、读取注册表项等。

在实际编程过程中，可以把开发的程序中的一些变量的值存储在注册表中，也可以从注册表中读取内容，让程序加以利用。因此注册表操作很有必要学习和研究。

VBA 操作注册表的方式有多种，本节介绍 VBA、VB6 的几个内置函数。

- ❑ GetSetting：根据指定的路径获取注册表项的值。
- ❑ GetSettings：与 GetSetting 类似，返回一个数组。
- ❑ SaveSetting：保存内容到注册表中。
- ❑ DeleteSetting：删除一个注册表项。

需要注意的是，上述 4 个内置函数的操作范围只限于 HKEY_CURRENT_USER\Software\VB and VBA Program Setting 这个键值内部。

按下快捷键【 Windows +R 】，输入 regedit，按回车键即可弹出注册表编辑器。然后依次单击节点 HKEY_CURRENT_USER\Software\VB and VBA Program Setting，就可以进入该文件夹中，如图 2-19 所示。

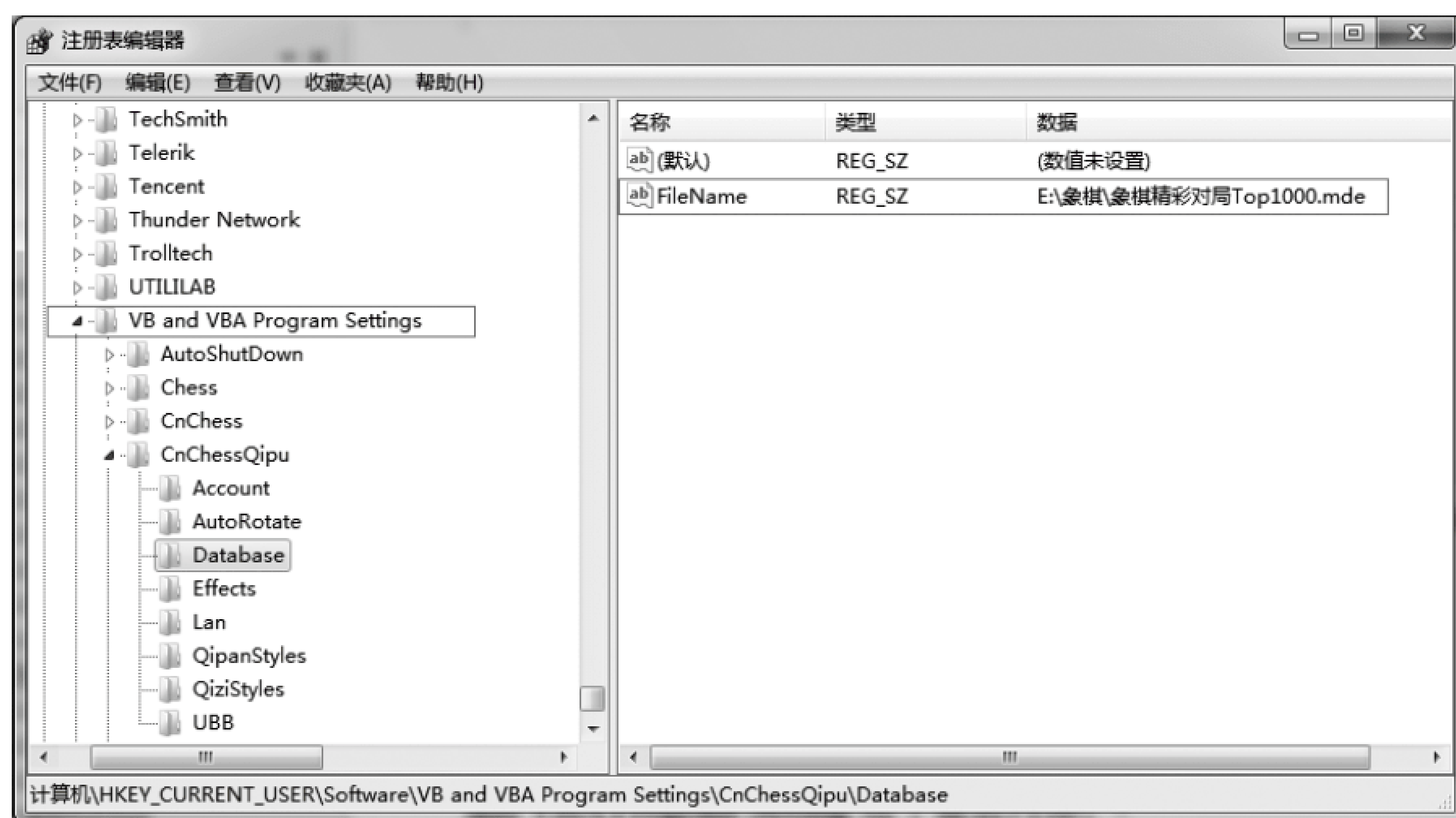


图 2-19　VB、VBA 专用的注册表项

以上四个内置函数的参数几乎都包含 AppName、Section、Key 三个层级。

以图中所选的节点为例，AppName、Section、Key 依次是 CnChessQipu、DataBase、FileName。

2.2.1　GetSetting

GetSetting 的作用是从指定的层级获取 Key 的属性值，返回一个字符串。

```
Sub ReadKey()
    Dim v As String
    v = GetSetting(AppName:="CnChessQipu", Section:="DataBase", Key:= "FileName")
    MsgBox v
End Sub
```

运行上述过程，对话框中返回注册表中 FileName 的属性值，如图 2-20 所示。

与注册表中对比，完全一致，如图 2-21 所示。

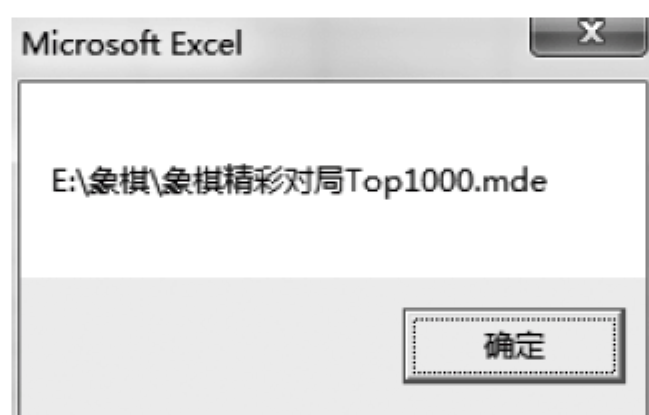


图 2-20　获取注册表信息



图 2-21　核对获取的结果

需要注意的是，如果 GetSetting 函数的三个参数中的任何一个写错，都将造成找不到注册表项，不会出现运行错误，而是返回一个空字符串。

2.2.2　SaveSetting

在使用 SaveSetting 函数的过程中，如果指定的层级存在，就修改现有层级的属性值；如果层级不存在，则会自动创建层级路径。

SaveSetting 是对注册表的修改，没有返回值，因此后面的参数不需要括号。

运行下面的过程，修改 FileName 这个 Key 的属性值。

```
Sub ModifyKey()
    Dim v As String
    v = "C:\temp\2018.mde"
    SaveSetting AppName:="CnChessQipu", Section:="DataBase", Key:="FileName",
Setting:=v
End Sub
```

刷新一下注册表，看到属性值已修改，如图 2-22 所示。

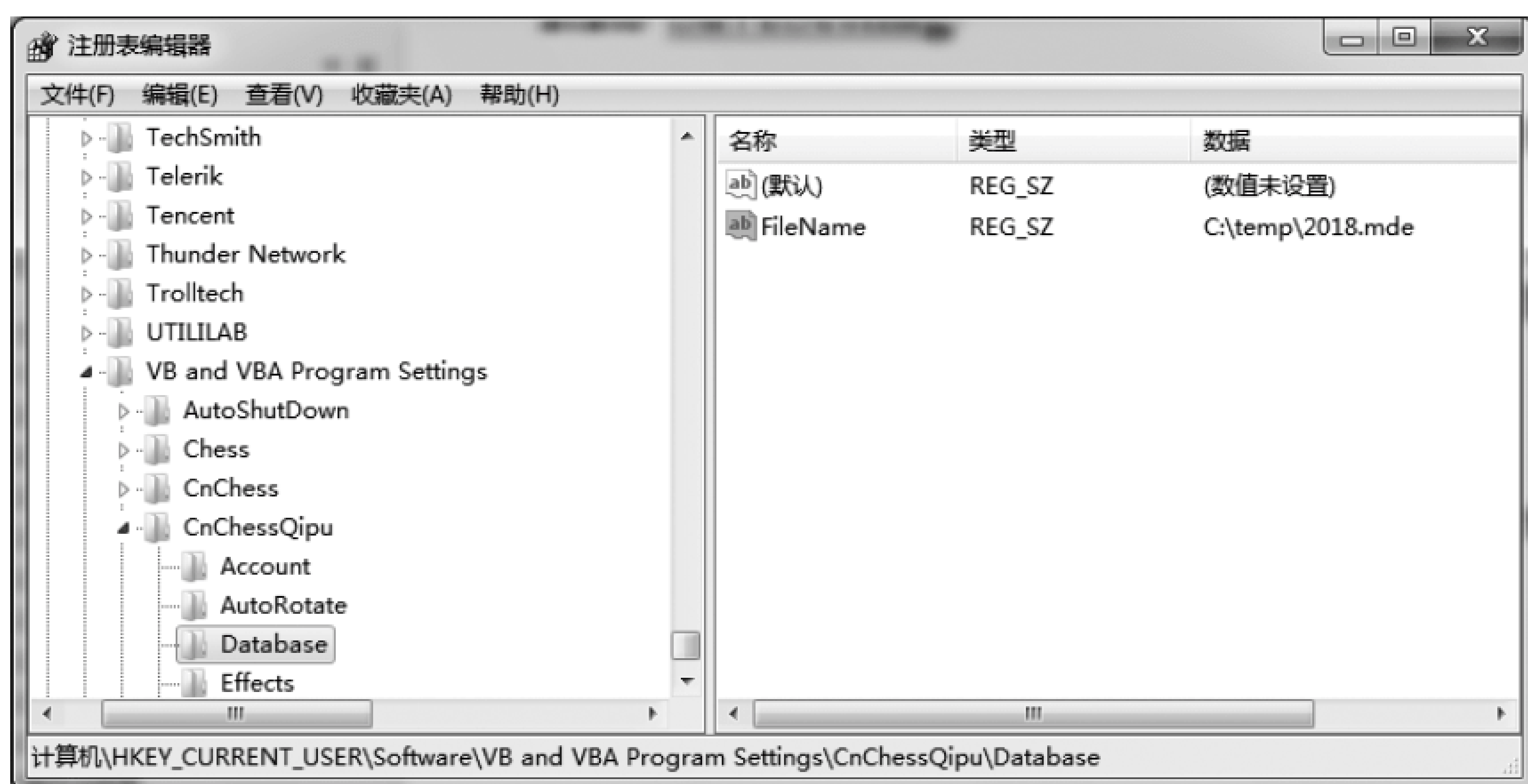


图 2-22　修改注册表信息

在编程开发过程中，经常需要在一个 Section 下面建立多个 Key，此时会对应多个属性值。下面以世界各国信息为例，说明一下注册表项的存取过程，如表 2-3 所示。

表 2-3　世界各国基本信息表

国　　家	首　　都	国土面积 / 万平方公里	人口 / 亿
中国	北京	960	13.75
美国	华盛顿	962	3.231
俄罗斯	莫斯科	1709	1.443
加拿大	渥太华	998	0.3628

假设要把表 2-3 中的内容保存到注册表中，那么“国家”这一列就相当于 Section，而首都、国土面积和人口都是 Key，要分别保存。以上所有项目都位于“世界大国”这个 AppName 下面。

```
Sub SaveKey()
    SaveSetting "世界大国", "中国", "首都", "北京"
    SaveSetting "世界大国", "中国", "国土面积", "960"
    SaveSetting "世界大国", "中国", "人口", "13.75"
    SaveSetting "世界大国", "美国", "首都", "华盛顿"
    SaveSetting "世界大国", "美国", "国土面积", "962"
    SaveSetting "世界大国", "美国", "人口", "3.231"
    SaveSetting "世界大国", "俄罗斯", "首都", "莫斯科"
    SaveSetting "世界大国", "俄罗斯", "国土面积", "1709"
    SaveSetting "世界大国", "俄罗斯", "人口", "1.443"
    SaveSetting "世界大国", "加拿大", "首都", "渥太华"
    SaveSetting "世界大国", "加拿大", "国土面积", "998"
    SaveSetting "世界大国", "加拿大", "人口", "0.3628"
End Sub
```

运行上述过程，刷新注册表，可以看到注册表中已存在相关信息，如图 2-23 所示。



图 2-23　批量存入注册表

2.2.3 DeleteSetting

DeleteSetting 用于删除一个注册表节点，这里的节点可以是 AppName、Section、Key 中的任何一个。

DeleteSetting 需要指定的参数也是三个，但是根据需要，可以更改参数个数。规定的参数越少，删除的层级越高，删除的内容也越多。

```
DeleteSetting "世界大国", "加拿大", "国土面积"
```

表示删除加拿大的国土面积这个 Key 和属性。

```
DeleteSetting "世界大国", "加拿大"
```

表示删除加拿大这个 Section，也就是删除加拿大这个文件夹。

```
DeleteSetting "世界大国"
```

将会删除整个 AppName 根节点。

2.2.4 GetAllSettings

获取注册表的方法，除了前面讲过的 GetSetting 函数以外，还可以用 GetAllSettings 函数一次性获取一个 Section 下面的所有 Key 及其属性值。获取到的内容是一个多行 2 列的二维数组。

下面的代码把“中国”这个 Section 的所有属性装载到字符串数组 v() 中。

```
Sub ReadAllKeys()
    Dim v() As String
    v = GetAllSettings(AppName:="世界大国", Section:="中国")
    Range("A1").Resize(UBound(v, 1) + 1, 2).Value = v
End Sub
```

单步执行上述过程，可以在本地窗口看到 v() 是一个 3 行 2 列的二维数组，如图 2-24 所示。

```
Sub ReadAllKeys()
    Dim v() As String
    v = GetAllSettings(AppName:="世界大国", Section:="中国")
    Range("A1").Resize(UBound(v, 1) + 1, 2).Value = v
End Sub
```

本地窗口

VBAProject.读写注册表.ReadAllKeys

表达式	值	类型
读写注册表		读写注册表/读写注册表
v		String(0 to 2, 0 to 1)
v(0)		String(0 to 1)
v(0,0)	"首都"	String
v(0,1)	"北京"	String
v(1)		String(0 to 1)
v(1,0)	"国土面积"	String
v(1,1)	"960"	String
v(2)		String(0 to 1)
v(2,0)	"人口"	String
v(2,1)	"13.75"	String

图 2-24 一次性获取全部注册表信息

为了把数组放到 Excel 单元格中，采用 Range 的 Resize 方法根据数组大小自动扩展区域。

运行上述过程后，表格单元格的结果如图 2-25 所示。

以上内容的源代码文件为“实例文档 07.xlsm”。

以上四个注册表操作函数一般用于产品开发时存取程序信息，如果要操作其他场所的注册表项，需要用到 2.3 节讲述的 WshShell 对象。

A1

	A	B
1	首都	北京
2	国土面积	960
3	人口	13.75

图 2-25　将注册表信息发送到单元格

2.3　使用 WshShell 操作注册表

WshShell 对象可以运行程序、操作注册表、创建快捷方式、访问系统文件夹、管理环境变量等。

要在 VBA 中使用该对象，需要向工程添加外部引用“ Windows Script Host Object Model”，如图 2-26 所示。

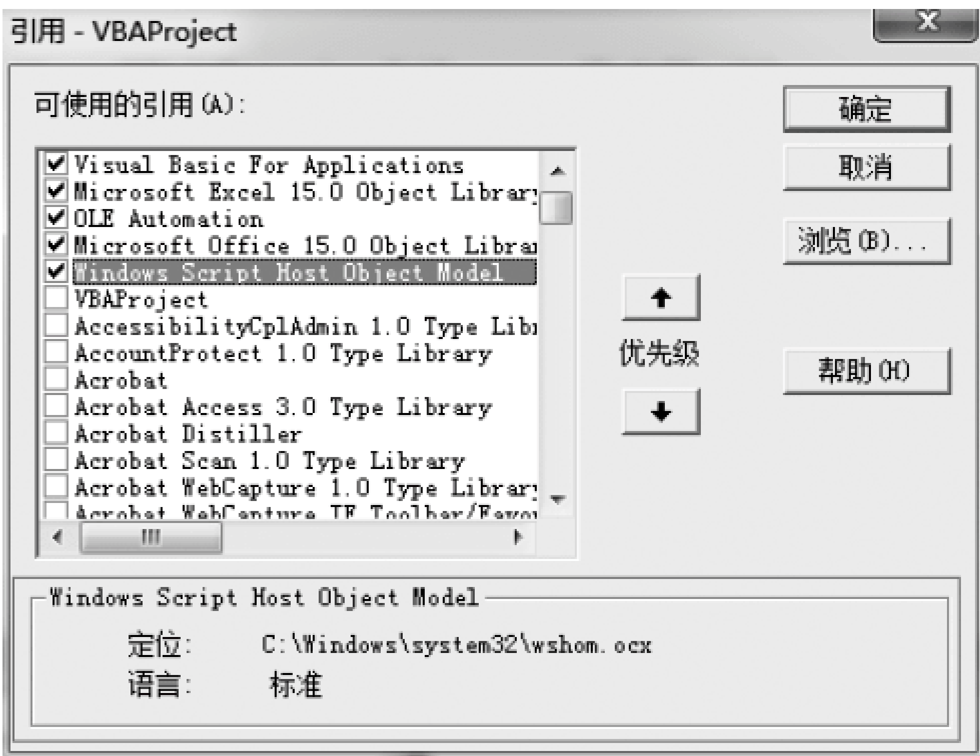


图 2-26　添加外部引用

代码中声明为：

```
Dim WS As New IWshRuntimeLibrary.WshShell
```

后期绑定方式为：

```
Set WS = CreateObject("WScript.Shell")
```

WshShell 对象中，用于操作注册表的函数有如下三个。

- RegRead：读注册表项。
- RegWrite：写注册表项。
- RegDelete：删除注册表项。

2.3.1　读注册表项

在使用 RegRead 函数时，只需要一个注册表项的完整路径即可返回注册表值。

Excel 2013 的宏安全性设置其实是保存在注册表中的，通过查看注册表编辑器，按照如下路径可以找到 Security 节点，如图 2-27 所示。

```
HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\Security
```

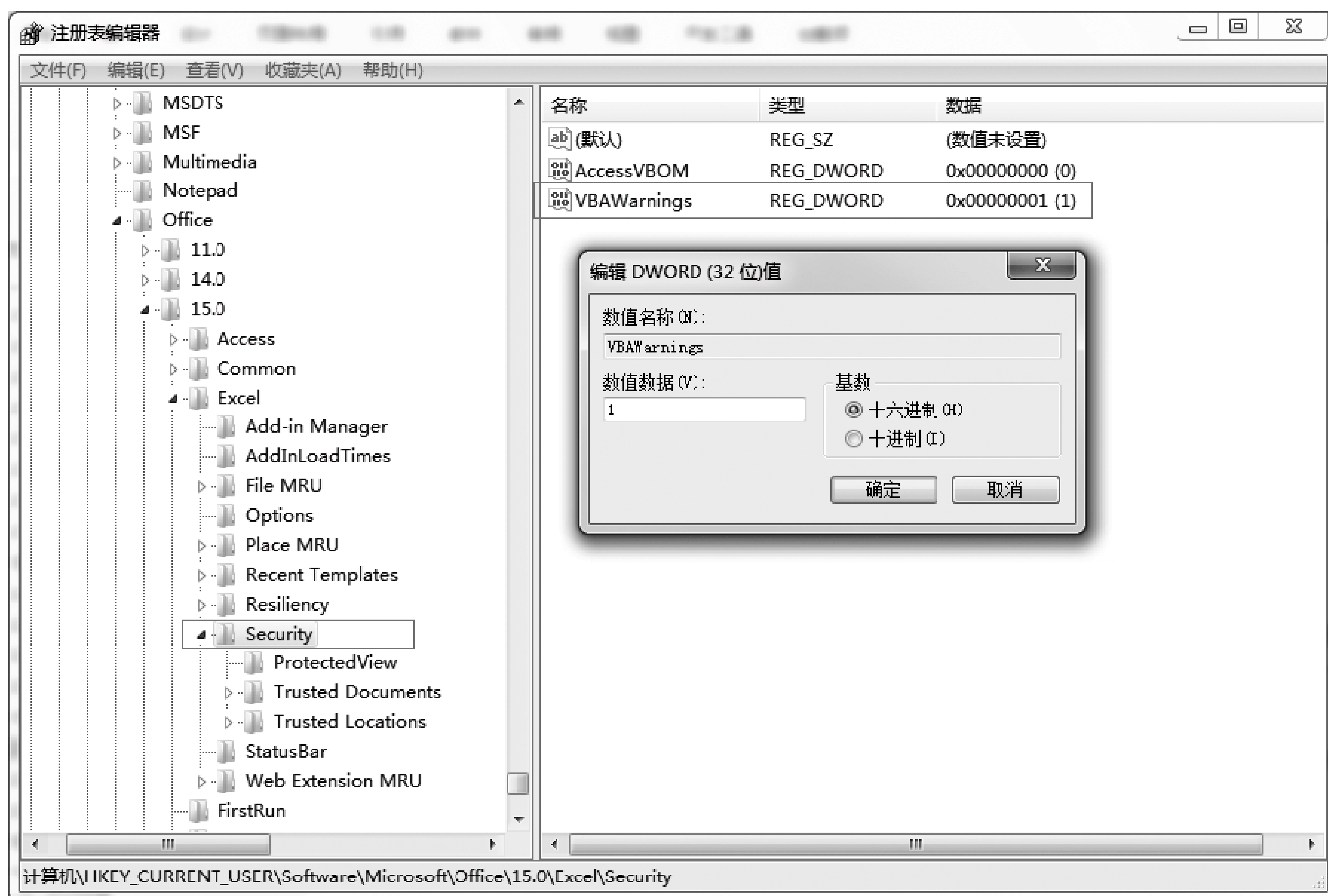


图 2-27 注册表信息

可以看到下面有一个 Key 为 VBAWarnings，Key 的取值对应于 Excel 宏安全性的设置（从上到下的 4 个单选按钮对应的属性值依次是 4、2、3、1），如图 2-28 所示。



图 2-28 VBA 安全性级别与注册表的对应关系

也就是说，手工在 Excel 中修改了宏安全性级别，注册表会同步变化；反之，通过注册表修改 VBAWarnings 的属性值，Excel 的宏安全性也同步发生变化。

下面的代码用来判断当前 Excel 的宏安全性。

```
Sub Test1()
    Dim WS As New IWshRuntimeLibrary.WshShell
    With WS
        MsgBox .RegRead("HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\
Security\VBAWarnings")
    End With
End Sub
```

以上代码运行后，返回一个十进制数 1，表示“启用所有宏”。

注意　如果 RegRead 函数中的注册表路径不存在，则会弹出“自动化”错误，如图 2-29 所示。

为了确保注册表路径书写不出问题，可以在注册表编辑器中右击节点，在弹出菜单中选择“复制项名称”命令，可以把完整路径复制到剪贴板，如图 2-30 所示。



图 2-29　注册表路径不存在引起的错误

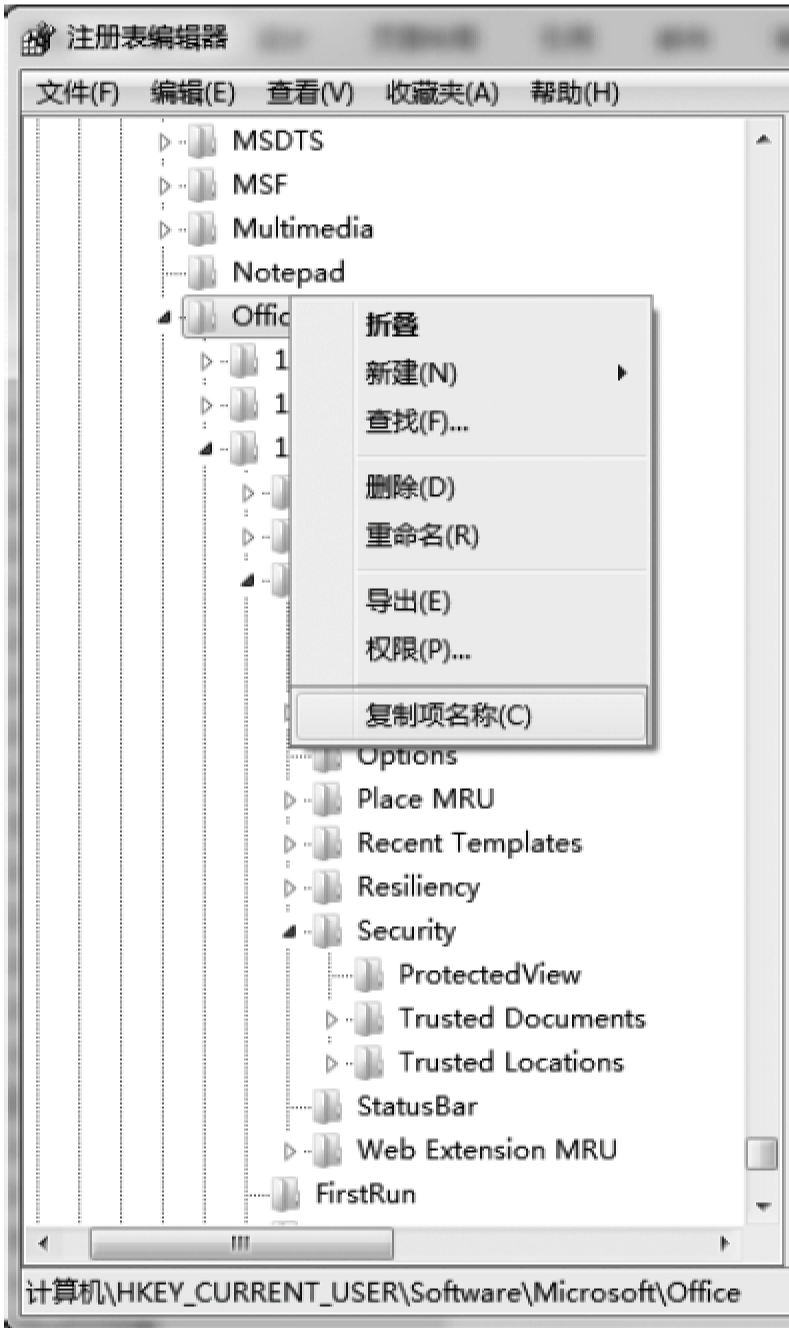


图 2-30　复制项名称

2.3.2　写注册表项

与 RegRead 函数相对应的是 RegWrite 方法，该方法用于修改注册表的值。

RegWrite 方法包含以下 3 个参数。

- ❑ Name：注册表的路径字符串。
- ❑ Value：要设定的值。
- ❑ Type：注册表项的类型，可以是 REG_SZ（字符串值）、REG_DWORD（DWORD 32 位置）、REG_BINARY（二进制值）。

下面的过程，通过改变注册表值，自动修改 Excel 2013 的宏安全性为“禁用所有宏，并发出通知”。

```
Sub Test2()
    Dim WS As New IWshRuntimeLibrary.WshShell
    With WS
        .RegWrite Name:="HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\
Security\VBAWarnings", Value:=2, Type:="REG_DWORD"
```

```
    End With
End Sub
```

代码分析：根据注册表编辑器，可以看到该注册表项的类型是一个 DWORD 值，因此 Type 参数设置为“REG_DWORD”。

2.3.3 删除注册表项

RegDelete 方法的语法非常简单，只需要规定注册表项的路径即可。

下面的代码删除注册表中的 Key VBAWarnings。

```
Sub Test3()
    Dim WS As New IWshRuntimeLibrary.WshShell
    With WS
        .RegDelete "HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\Security\
VBAWarnings"
    End With
End Sub
```

实际上，注册表和资源管理器类似，也是一个树状结构，严格地讲，RegDelete 方法不仅可以删除 Key，还可以删除各层级的文件夹。也就是说：

```
RegDelete "HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\Security\"
```

理论上可以删除 Security 整个文件夹，但微软不允许用户删除，因此运行时可能会出错。

最后演示一个用代码自动读取、设置记事本的字体名称。Notepad 是 Windows 系统的默认程序，属于微软开发的产品。在注册表中查看该节点，可以看到右侧有大量的 Key，这些其实就是记事本的配置信息，如图 2-31 所示。

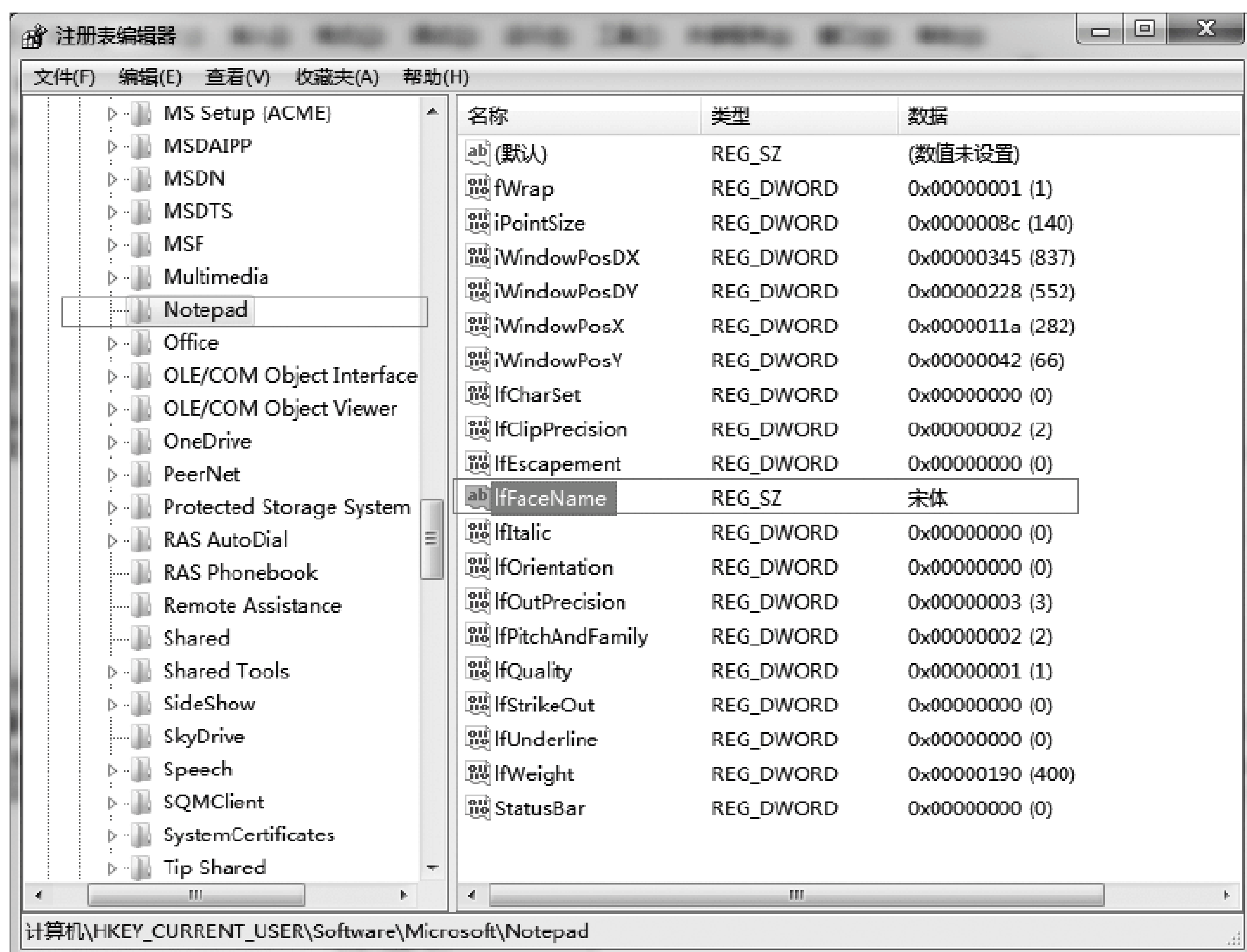


图 2-31 记事本程序的注册表信息

可以看到字体名称的 Key 是 lfFaceName，这是一个 REG_SZ 字符串，当前字体为“宋体”。

下面的过程首先读取记事本程序的字体名称，然后用 RegWrite 方法设置为“华文仿宋”。

```
Sub Test4()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim fontname As String
    With WS
        fontname = .RegRead("HKEY_CURRENT_USER\Software\Microsoft\Notepad\lfFaceName")
        MsgBox "记事本程序现在的字体是: " & fontname
          .RegWrite Name:="HKEY_CURRENT_USER\
Software\Microsoft\Notepad\lfFaceName", Value:="华文
仿宋", Type:="REG_SZ"
    End With
End Sub
```

运行上述代码后，再次打开记事本程序，可以看到字体风格已改变，如图 2-32 所示。

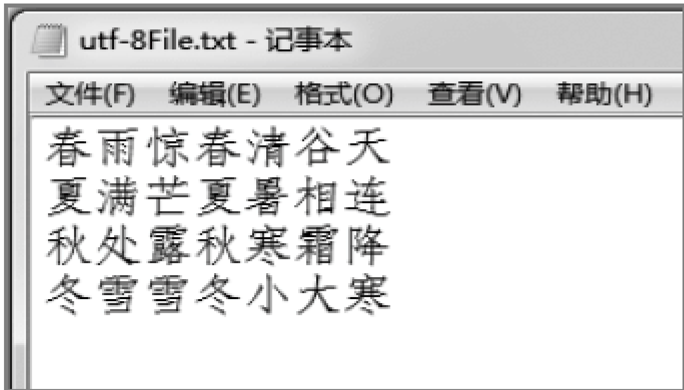


图 2-32　修改注册表，从而修改记事本程序的字体设置

2.3.4　创建新项

在编程过程中，除了修改现有项目外，很多情况下需要创建新的子项。

例　如 HKEY_CURRENT_USER\Software\Microsoft\Office\Excel\Addins\customUI_Excel 这个注册表项类似于资源管理器中的文件夹，该注册表项的本身属性是一个字符串：“ By Ryueifu”，此外，该注册表项还包括 4 个属性子项。

- ❑ Description 字符串子项：属性值是 CustomUI_Excel。
- ❑ FriendlyName 字符串子项：属性值是 CustomUI_Excel。
- ❑ LoadBehavior 整数子项：属性值是 2。
- ❑ Manifest 字符串子项：属性值是一个路径文本。

右击注册表项，可以在弹出菜单中为已有子项的注册表项添加新的子项，如图 2-33 所示。

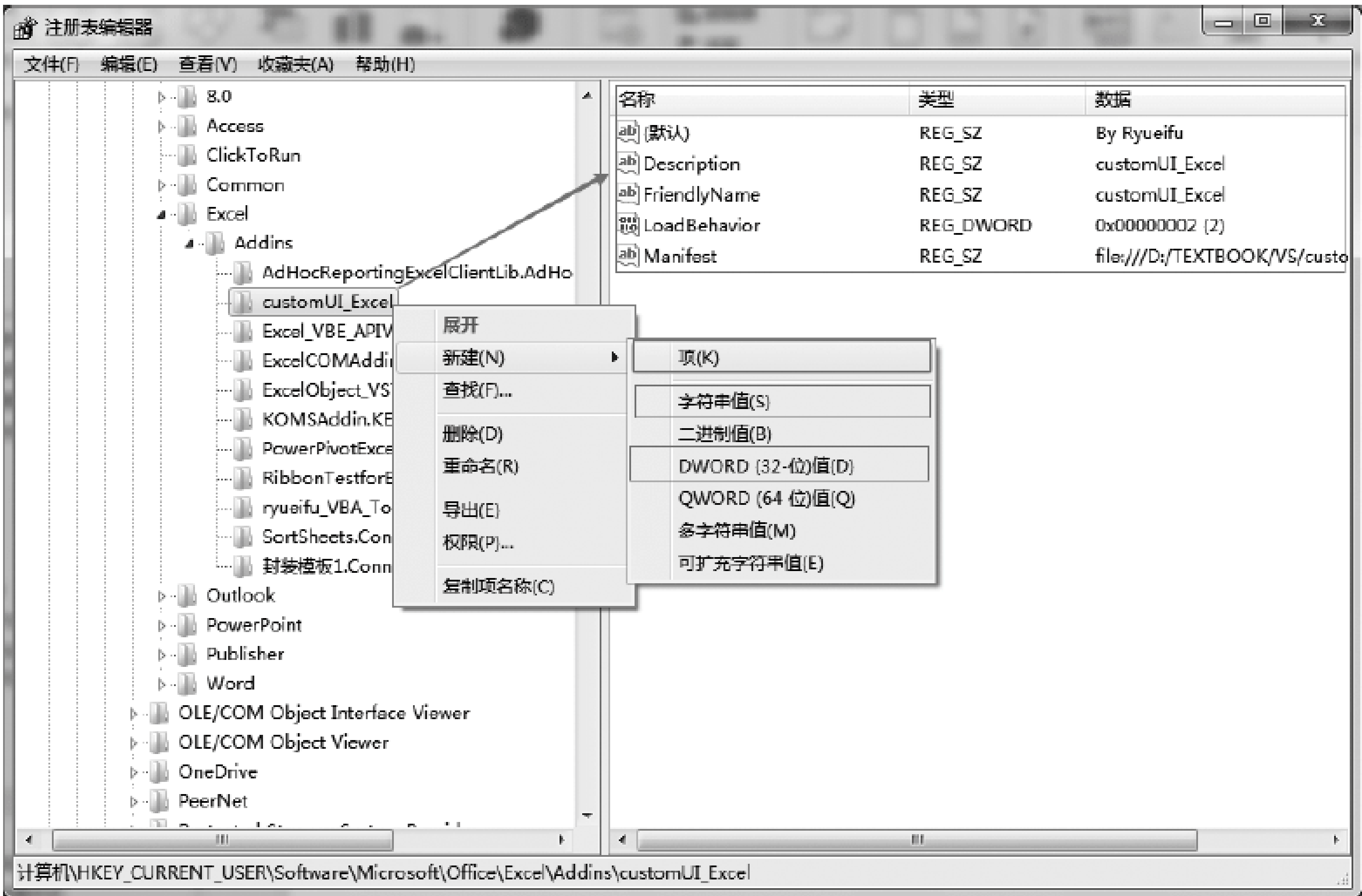


图 2-33　为注册表创建新项

在注册表中右击 customUI_Excel 注册表项，在弹出菜单中可以新建属性子项，也可以新建类似于文件夹一样的目录子项。

使用 VBA 同样可以创建新项。下面的过程包含 4 个修改注册表命令，其中前两个修改现有子项的数值，第 3 个命令增加一个名为 Version 的属性子项，第 4 个命令增加一个名为 User 的目录子项。

```
    Sub Test5()
        Const parent As String = "HKEY_CURRENT_USER\Software\Microsoft\Office\Excel\
Addins\"
        Dim WS As New IWshRuntimeLibrary.WshShell
        With WS
            .RegWrite Name:=parent & "customUI_Excel\", Value:="Ryukou" '修改本身属性
            .RegWrite Name:=parent & "customUI_Excel\Description", Value:="对插件的
描述信息", Type:="REG_SZ"                              '修改已有属性值
            .RegWrite Name:=parent & "customUI_Excel\Version", Value:="2", Type:=
"REG_DWORD"                                            '创建属性子项 Version=2
            .RegWrite Name:=parent & "customUI_Excel\User\", Value:="注册表小白"
                                                       '创建子文件夹，并赋予本身属性值
        End With
    End Sub
```

运行上述过程，customUI_Excel 的注册表项发生变化，如图 2-34 所示。

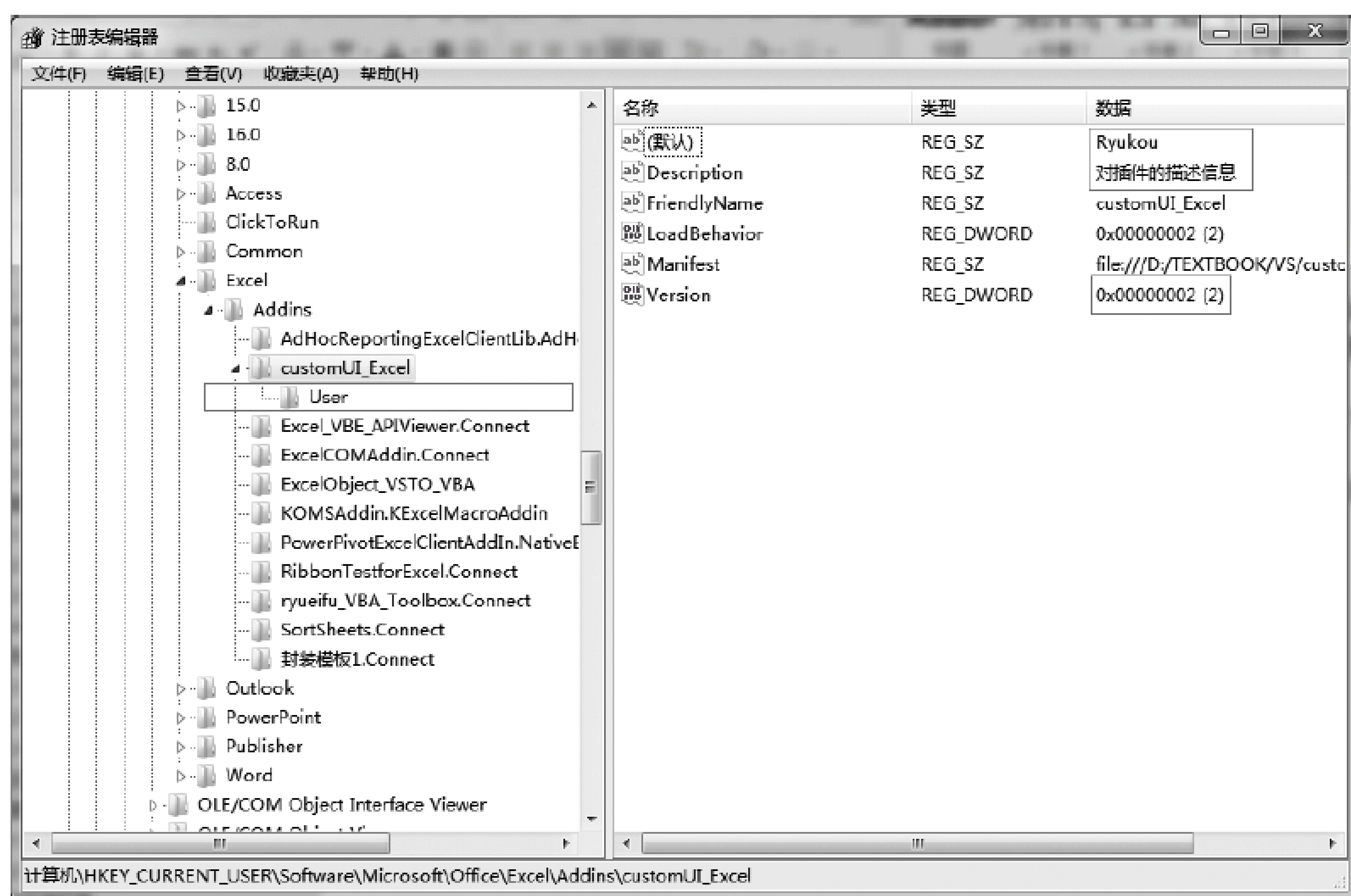


图 2-34 使用代码自动修改注册表

需要特别注意的是，注册表路径末尾是否添加反斜杠，对操作后的影响非常大，例如 RegWrite Name:=parent & "customUI_Excel\", Value:="Ryukou" 这一句的含义是修改 customUI_Excel 的默认属性为 Ryukou。

如果写成 RegWrite Name:=parent & "customUI_Excel", Value:="Ryukou" 则表示在 Addins 这个注册表项下面创建一个名为 customUI_Excel 的属性子项！

因此，在 VBA 中使用 RegRead、RegWrite 和 RegDelete 方法对注册表项进行操作时，一定要思考加与不加反斜杠的区别。

以上内容的源代码文件为“实例文档 08.xlsm”。

2.4　创建快捷方式

Windows 允许用户在文件夹中或者桌面（桌面是系统盘下的一个特殊文件夹）为文件或网址创建快捷方式。所谓的快捷方式，其实是一个扩展名为 .lnk 的图标文件。

在桌面上查看搜狗高速浏览器的快捷方式的属性，可以看到这个快捷方式指向的文件是 SogouExplorer.exe，也就是说，双击快捷方式就相当于双击了搜狗高速浏览器的执行文件，如图 2-35 所示。

然后切换到“详细信息”选项卡，可以看到该快捷方式的完整路径，如图 2-36 所示。



图 2-35　搜狗高速浏览器的桌面快捷方式

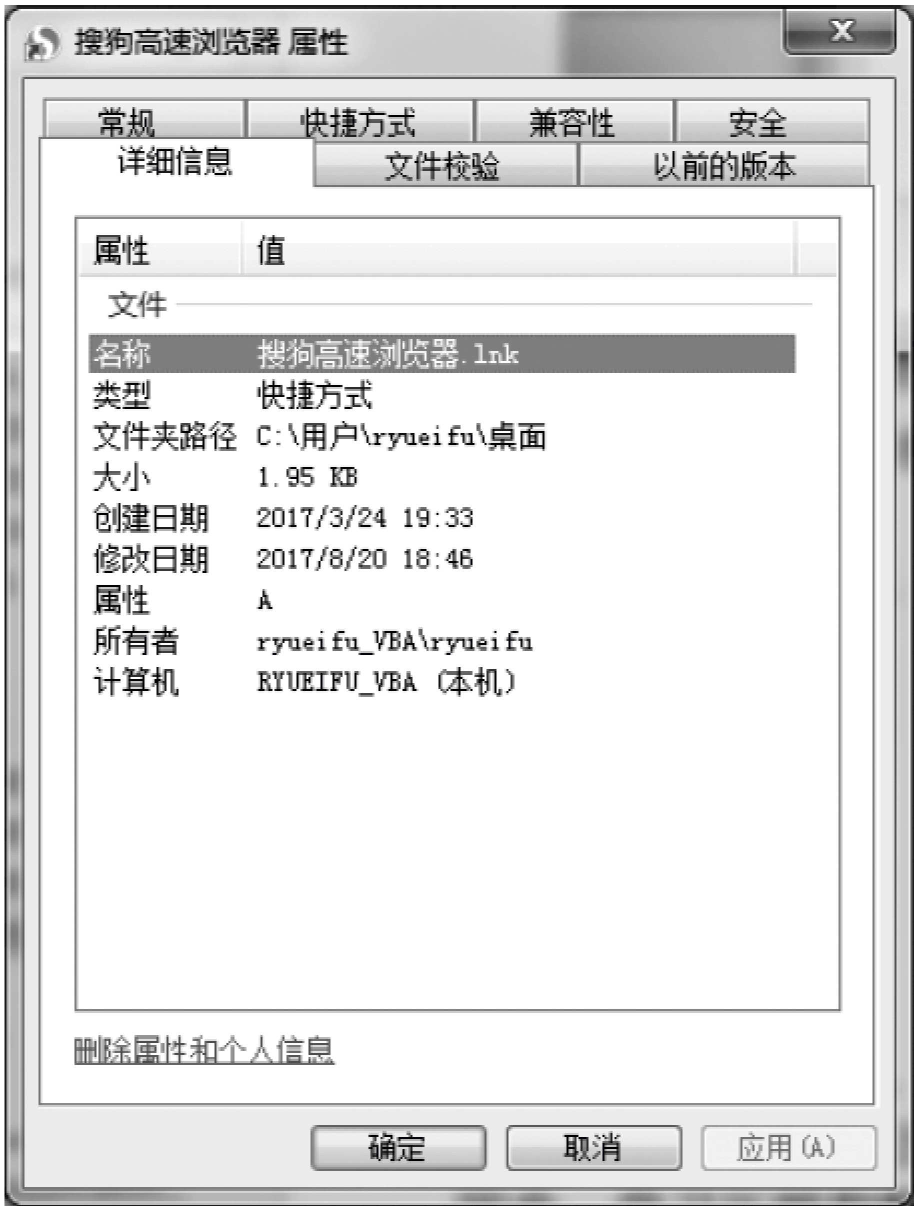


图 2-36　快捷方式的详细信息

看到所在路径是桌面，文件名是“搜狗高速浏览器 .lnk”。

2.4.1　创建文件的快捷方式

创建快捷方式也可以通过编程的方式实现。具体步骤如下。

首先，使用 WshShell 的 CreateShortCut 方法创建一个快捷方式，会产生一个 WshShortcut 对象，然后，为该对象设置有关属性。

- ❑ TargetPath：目标路径，也就是快捷方式指向的文件路径。
- ❑ IconLocation：自定义图标的位置，也就是 .ico 格式的图标文件位置。
- ❑ HotKey：快捷键。

最后，保存快捷键。

当然，在创建快捷方式之前应该先判断桌面是否已经有了同名的快捷方式，如果已存

在，会创建失败。

下面的代码首先判断桌面上是否存在 Windows 计算器的快捷方式，如果已有快捷方式，则弹出警告对话框，不予创建。否则，自动在桌面创建一个名为 calc.lnk 的快捷方式。

```
Sub CreateShortCut()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim FSO As New IWshRuntimeLibrary.FileSystemObject
    Dim Desk As String
    Dim mySht As IWshRuntimeLibrary.WshShortcut
    Desk = WS.SpecialFolders("Desktop")
    If FSO.FileExists(Desk & "\calc.lnk") Then
        MsgBox "已存在桌面快捷方式，拒绝创建！ ", vbCritical
    Else
        Set mySht = WS.CreateShortCut(Desk & "\calc.lnk")
        With mySht
            .TargetPath = "C:\Windows\System32\calc.exe"
            .IconLocation = "C:\temp\WN.ICO"
            .Hotkey = "Ctrl+Alt+F7"
            .Save
        End With
    End If
End Sub
```

代码分析：变量 Desk 用来获取桌面所在的文件夹路径，对象变量 mySht 就是快捷方式本身。读者可以根据需要自行调整 TargetPath、IconLocation 这些参数。

运行上述代码后，可以看到桌面上多了一个快捷方式。图 2-37 所示为 calc 快捷方式的属性。



图 2-37 自动创建桌面快捷方式

如果双击该快捷方式，或者按下快捷键【Ctrl+Alt+F7】，会自动弹出计算器。

2.4.2　创建网址的快捷方式

除了可以创建本机文件的快捷方式，还可以创建网址的快捷方式，只要双击该快捷方式，就自动在默认的浏览器中打开该网页。

网址快捷方式与上面介绍过的普通快捷方式有几点不同。

- 对象类型不同，网址快捷方式的对象类型是 WshURLShortcut。
- 快捷方式扩展名不同，网址快捷方式的扩展名是 .url，不是 .lnk。
- 没有 HotKey 属性，不支持快捷键的设定。

下面的过程自动在桌面创建一个网址快捷方式。

```
Sub CreateURLShortCut()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim FSO As New IWshRuntimeLibrary.FileSystemObject
    Dim Desk As String
    Dim mySht As IWshRuntimeLibrary.WshURLShortcut
    Desk = WS.SpecialFolders("Desktop")
    If FSO.FileExists(Desk & "\MaHouPao.url") Then
        FSO.DeleteFile Desk & "\MaHouPao.url"
    End If
    Set mySht = WS.CreateShortCut(Desk & "\MaHouPao.url")
    With mySht
        .TargetPath = "http://vba.mahoupao.net/forum.php"
        .Save
    End With
End Sub
```

代码分析：首先判断桌面是否已存在该网址快捷方式，如果存在，则先删除。

运行上述代码，会看到桌面多了一个快捷方式。图 2-38 所示为创建的 MaHouPao 快捷方式的属性。



图 2-38　网址快捷方式属性

2.5 操作环境变量

环境变量（Environment Variables）一般是指指定操作系统运行环境的一些参数，如临时文件夹位置和系统文件夹位置等。

环境变量是在操作系统中一个具有特定名字的对象，它包含了一个或者多个应用程序所将用到的信息。例如 Windows 和 DOS 操作系统中的 path 环境变量，当要求系统运行一个程序而没有告诉它程序所在的完整路径时，系统除了在当前目录下面寻找此程序外，还应到 path 中指定的路径去找。例如之前讲过的 Shell 函数中，可执行文件的路径可以不写前面的所在路径，只写文件名称即可。这是因为这些文件的所在路径已经保存在环境变量中。

本节讲述如何用 WshShell 操作环境变量。

对于 Windows 7 系统，查看环境变量的方法是，进入控制面板，单击“系统和安全”→“系统”→“高级系统设置”，然后在“系统属性”对话框中切换到“高级”选项卡，接着单击“环境变量”按钮，如图 2-39 所示。



图 2-39 设置环境变量

然后弹出“环境变量”对话框，如图 2-40 所示。

对话框分为上下两部分，上面部分是用户变量（User），下面是系统变量（System），很多应用程序都把重要的路径保存在系统变量的 Path 变量中。

以系统变量为例，对话框中看到的 OS，就是一个环境变量的变量名称，而后面的 Windows_NT 是一个字符串，它表示变量的值。在逻辑上，与程序语言中的字典（键值对）非常相似。



图 2-40 “环境变量”对话框

通过环境变量对话框可以进行增加、删除、修改变量

等操作。

WshShell 对象下面有一个 Environment 成员，该成员会返回一个环境变量集合 WshEnvironment 对象，通过访问该对象，可以实现环境变量的查看、修改和增删。

2.5.1　查看和遍历环境变量

下面的过程遍历系统环境变量（System）的所有变量。

```
Sub Test1()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim SysVariables As IWshRuntimeLibrary.WshEnvironment
    Dim v
    Set SysVariables = WS.Environment("System")
    With SysVariables
        Debug.Print "变量数量为", .Length
        For Each v In SysVariables
            Debug.Print v
        Next v
    End With
End Sub
```

代码分析：针对遍历到的每一个变量，v 得到的是一个字符串，对于每一个字符串，以 = 为分界线，等号左侧是变量名，右侧是变量值，如图 2-41 所示。

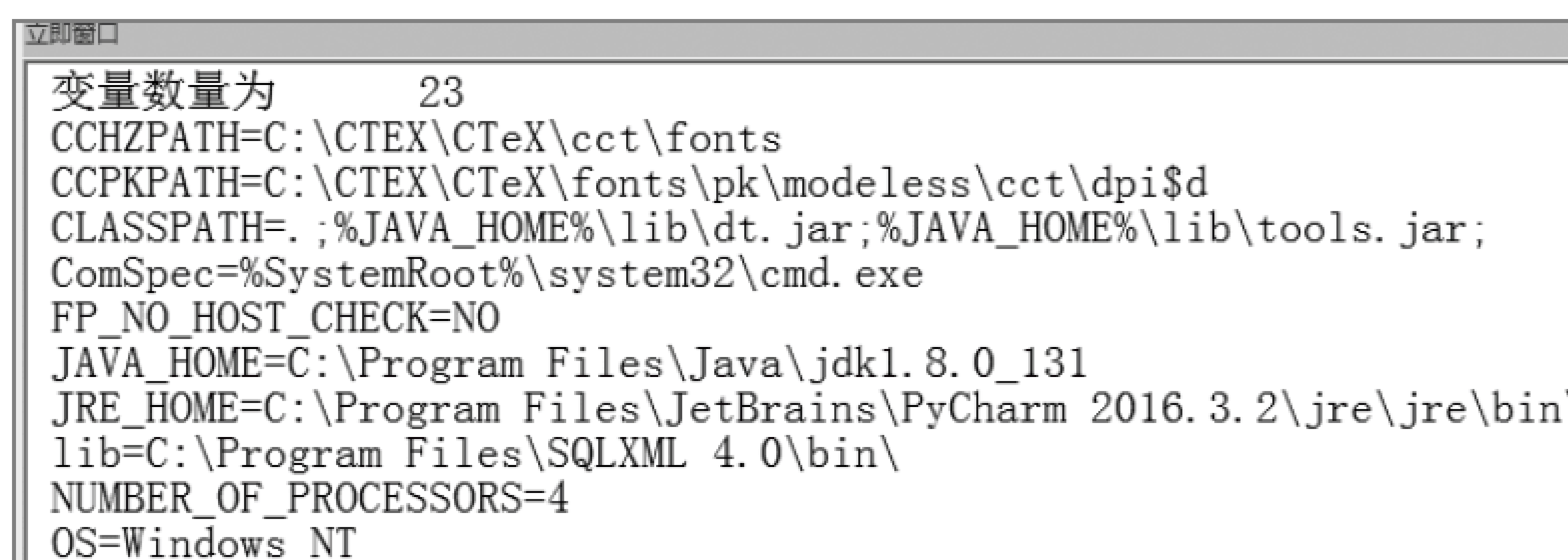


图 2-41　遍历系统环境变量

如果要单独显示变量名和字符串，用 Split 函数处理一下即可。也就是把打印语句改为如下形式。

```
Debug.Print Split(v, "=")(0), Split(v, "=")(1)
```

如果要查看其中某个环境变量的值，可以用 Item 获取。

运行下面的过程，获取 CCHZPATH 这个环境变量的取值。

```
Sub Test2()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim SysVariables As IWshRuntimeLibrary.WshEnvironment
    Dim v
    Set SysVariables = WS.Environment("System")
    MsgBox SysVariables.Item("CCHZPATH"), vbInformation, "CCHZPATH"
End Sub
```

运行上述过程，对话框中给出环境变量的值，如图 2-42 所示。

如果要遍历用户环境变量，只需要把上述过程中的 WS.Environment("System") 替换成 WS.Environment("User") 即可。

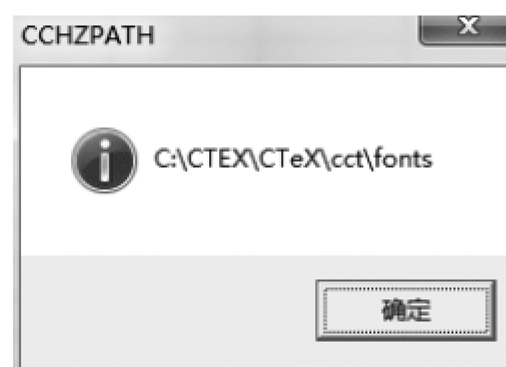


图 2-42　获取环境变量的值

2.5.2　新建和修改环境变量

无论是新建一个环境变量，还是修改已有变量的取值，语法都是一样的。

下面的过程更改 CCHZPATH 环境变量的路径。

```
    Sub Test2()
        Dim WS As New IWshRuntimeLibrary.WshShell
        Dim SysVariables As IWshRuntimeLibrary.
WshEnvironment
        Dim v
        Set SysVariables = WS.Environment ("System")
        SysVariables.Item("CCHZPATH") = "E:\
VBA;F:\DownLoad"
    End Sub
```

运行上述过程，然后在环境变量对话框中核对，如图 2-43 所示。

再例如 SysVariables.Item("Perfect") = "C:\temp" 可以直接创建一个名称为 Perfect 的新环境变量，并且赋值。



图 2-43　自动修改环境变量的值

2.5.3　删除环境变量

使用环境变量对象的 Remove 方法，可以删除指定名称的环境变量。例如 SysVariables.Remove "CCHZPATH" 把名为 "CCHZPATH" 的环境变量删除。

如果要删除所有环境变量，循环调用 Remove 方法即可。下面的代码删除 User 下的所有环境变量。

```
    Sub Test3()
        Dim WS As New IWshRuntimeLibrary.WshShell
        Dim SysVariables As IWshRuntimeLibrary.WshEnvironment
        Dim v
        Dim Col As New Collection
        Dim c
        Set SysVariables = WS.Environment("User")
        For Each v In SysVariables
            Col.Add Split(v, "=")(0)
        Next v
        For Each c In Col
            SysVariables.Remove CStr(c)
        Next c
    End Sub
```

代码分析：如果在循环过程中对集合中的元素进行增删，这样的操作比较危险。因此，首先把目前所有的环境变量的名称保存到一个 Collection 对象或者动态数组中。

然后循环 Collection 中的每一个字符串，再移除环境变量，如图 2-44 所示。



图 2-44　批量删除所有环境变量

2.6　自动激活指定标题文字的窗口

WshShell 对象下面的 AppActivate 函数可以激活屏幕上与 Office、VBA 不相关的窗口。其语法是：

```
AppActivate(App,Wait)
```

返回一个布尔值，找到窗口并激活，返回 True，否则返回 False。

参数 App 表示一个窗口的标题文字，是字符串。

参数 Wait 是一个布尔值，设置为 True 时，表示等待，也就是当激活完成后，才继续执行后面的代码。

假设桌面上启动了记事本，其窗口的标题文字为“无标题 - 记事本”，然后在工作表上插入一个图片或按钮，指定宏到 ActivateNotepad。具体代码如下。

```
Sub ActivateNotepad()
    Dim WS As New WshShell
    If WS.AppActivate(App:="无标题 - 记事本", Wait:=True) Then
        Debug.Print "激活成功"
    Else
        Debug.Print "激活失败"
    End If
End Sub
```

当单击工作表上的按钮时，记事本窗口自动弹出到最前，取得焦点，并且在立即窗口打印出“激活成功”，如图 2-45 所示。

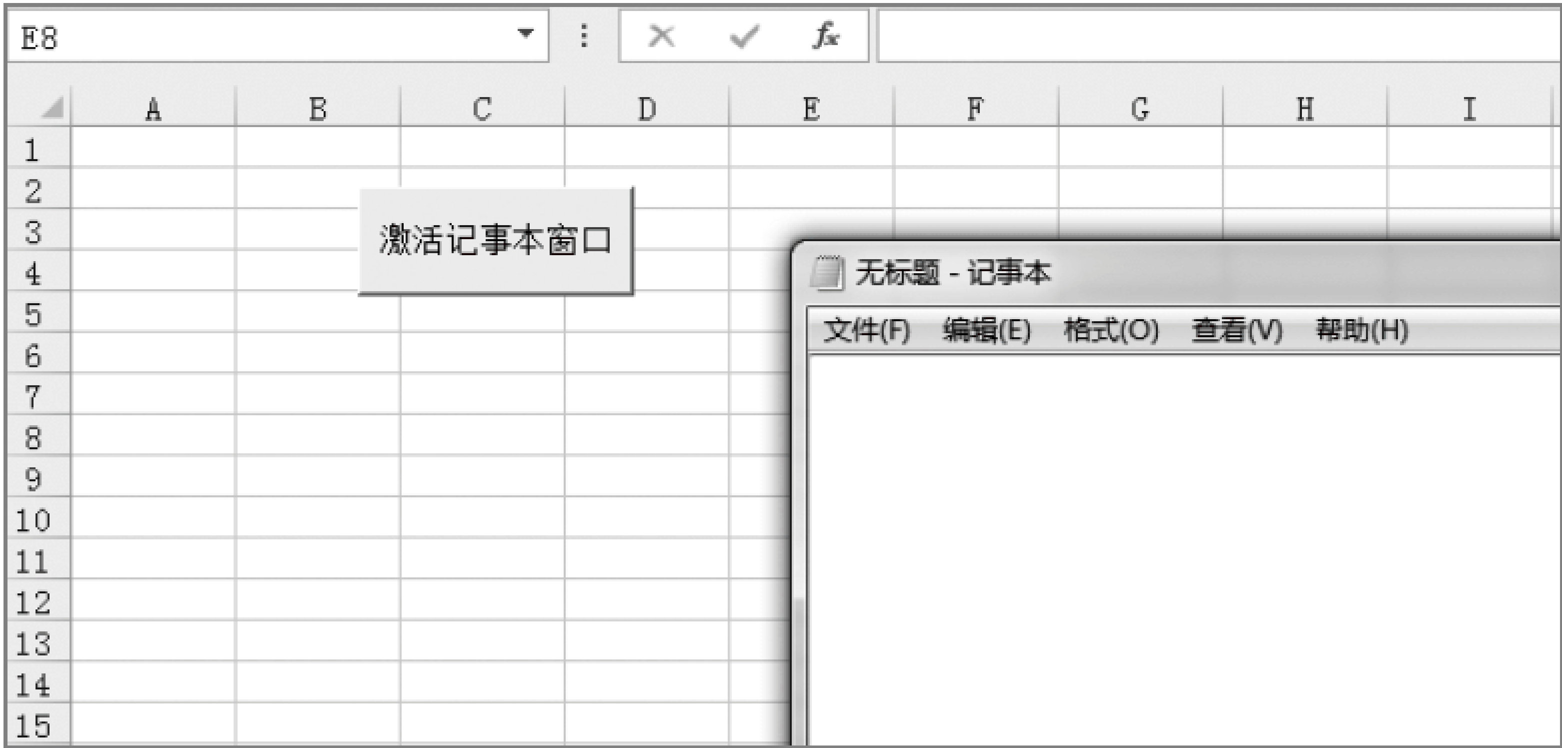


图 2-45　自动激活指定标题的窗口

注意 标题文字必须一字不差，如果记事本已经关闭，或者 App 参数中标题文字有误，都会导致 AppActivate 函数返回 False。

2.7 自动关闭的对话框

VBA 中的 MsgBox 对话框主要有两个作用，一个是在对话框中弹出运行结果，用于告知执行程序的人员；另一个作用是对话框中具有“是”“否”“取消”等可选按钮，让用户抉择。

但是 Msgbox 对话框一旦弹出来，用户必须与之交互，手工把对话框关掉方可执行后续的程序代码。

在很多情况下，需要用到 MsgBox 对话框的效果，如果能够在一定时间范围内自动关闭，就更为理想了。

WshShell 的 Popup 方法可以弹出一个对话框，理论上可以在设定的秒数之后自动关闭，但在实际运用中经常不能自动关闭。

因此，更推荐使用 API 函数来设计自动关闭的对话框。MsgBoxTimeout 的完整声明如下。

```
    Declare Function MsgBoxTimeout Lib "user32" Alias "MessageBoxTimeoutA" (ByVal
hwnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As
VbMsgBoxStyle, ByVal wlange As Long, ByVal dwTimeout As Long) As Long
```

参数说明如下。

- ❑ hwnd：对话框依附的窗口对象的句柄值。
- ❑ lpText：对话框中显示的内容。
- ❑ lpCaption：对话框的标题。
- ❑ wType：对话框显示的按钮、图标风格的组合值。
- ❑ wlange：函数扩展。
- ❑ dwTimeout：对话框持续的最长毫秒数。

MsgBoxTimeout 对话框弹出来后，如果用户主动单击对话框中的按钮，则对话框在设定时间之前就提前关闭，此时该函数返回的整型值与用户所选按钮相关，若置之不理，则对话框在规定时间过后自动关闭，自动关闭了的对话框，其返回值为 32000。

下面的程序在对话框中依次弹出一些判断题，用户可以选择“是”或“否”进行作答，如果没来得及单击按钮，则对话框在 10 秒后自动弹出下一道题。

程序代码如下。

```
    #If Win64 Then '64 位
        Private Declare PtrSafe Function MsgBoxTimeout Lib "user32" Alias "Message
BoxTimeoutA" (ByVal hwnd As Long, ByVal lpText As String, ByVal lpCaption As String,
ByVal wType As VbMsgBoxStyle, ByVal wlange As Long, ByVal dwTimeout As Long) As Long
    #Else
```

```
    Private Declare Function MsgBoxTimeout Lib "user32" Alias "MessageBoxTimeoutA"
(ByVal hwnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As
VbMsgBoxStyle, ByVal wlange As Long, ByVal dwTimeout As Long) As Long

#End If

Sub 开始答题 ()
    Dim i As Integer
    Dim L As Long
    For i = 2 To 6
        L = MsgBoxTimeout(hwnd:=0, lpText:=Range("A" & i).Value, lpCaption:=" 判断
题 ", wType:=vbYesNo + vbInformation, wlange:=1, dwTimeout:=10000)
        Select Case L
        Case 32000
            Range("B" & i).Value = " 未做 "
        Case vbYes
            Range("B" & i).Value = " 是 "
        Case vbNo
            Range("B" & i).Value = " 否 "
        End Select
    Next i
End Sub
```

代码分析：hwnd:=0 表示该对话框的宿主窗口是计算机的屏幕（屏幕的句柄是 0），如果设置为 hwnd:=Application.hwnd，则对话框的宿主是 Excel 应用程序，那么在对话框存续期间用户不能对工作表和单元格进行任何操作，有点类似模态窗体。

运行上述程序，弹出的对话框依次显示单元格中的每道题，如果用户动作慢没有来得及选择，则在 B 列标记为“未做”，如图 2-46 所示。

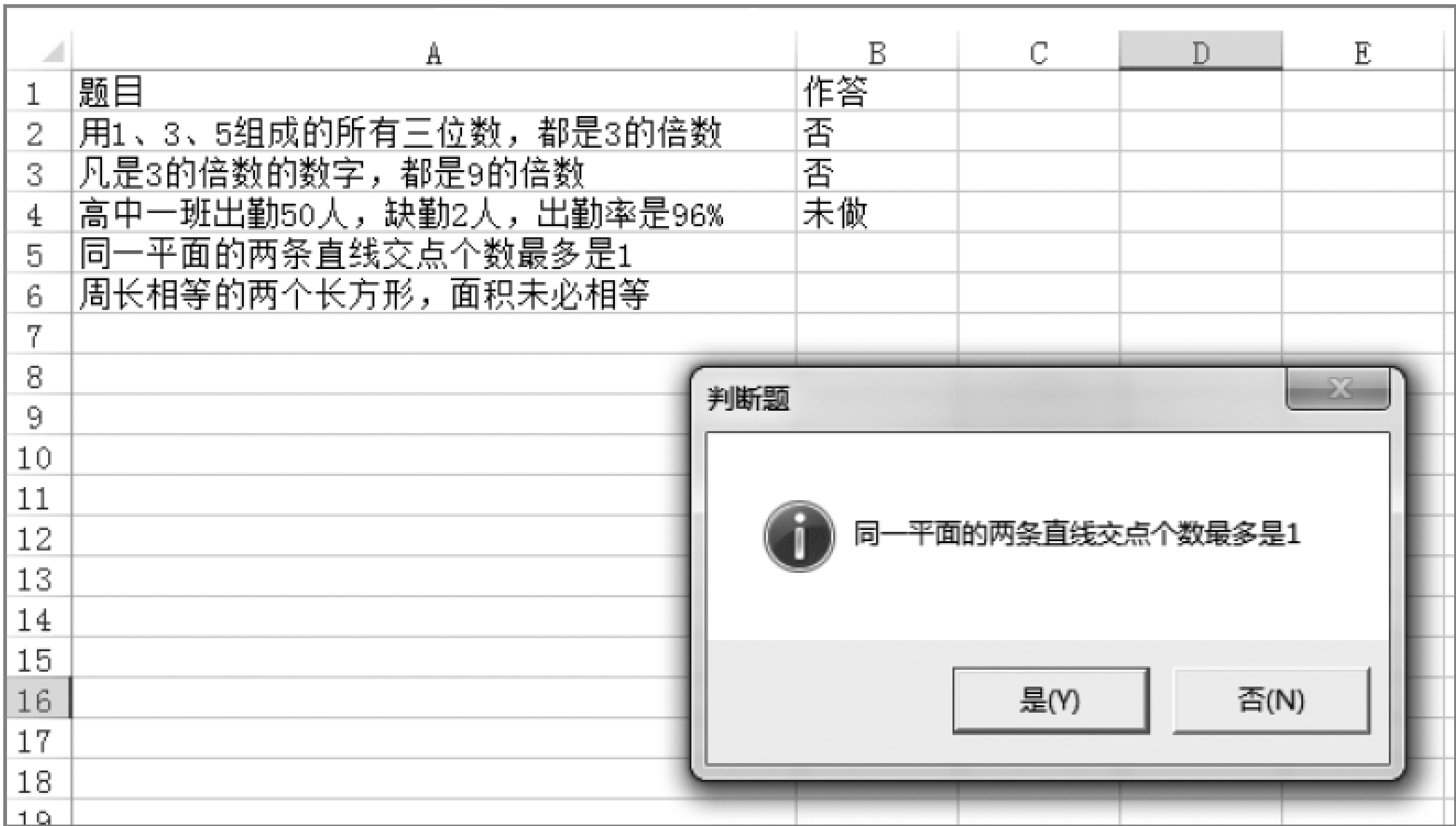


图 2-46 指定时间范围内自动关闭的对话框

以上程序的源代码文件为“自动关闭的对话框 .xlsm”。

2.8 自动发送按键

日常办公过程中，计算机的绝大多数操作都是通过操纵鼠标和键盘完成的。对于 Office

文档的处理，使用 VBA 编程可以很好地完成，但是如果要处理屏幕上不属于 Office 管辖的窗口，就需要自动按键、自动单击鼠标的技术。

关于自动单击鼠标，需要用到 API 技术，本书暂不作介绍。

虽然在 Excel VBA 中 Application 对象也有 Sendkeys 方法，但是仅限于 Excel VBA，Office 的其他组件（PowerPoint、Word 等）的 VBA 模型中并未提供这个方法。因此，本节讲述更加通用的 WshShell 的 Sendkeys 方法以实现自动按键。

熟练掌握自动按键，可以在非 Office 窗口中实现类似于 VBA 操作的效果，更大程度地减少手工操作。

2.8.1 按键写法

WshShell 的 Sendkeys 方法的功能就是在活动窗口中，按下指定的键（键盘上的键）。这里所说的活动窗口就是屏幕上置于最前的、具有焦点的一个窗口，而并非其代码宿主程序。

Sendkeys 语法很简单，然而重点和难点在于按键字符串的构造。因此首先列出键和代码的对应关系，如表 2-4 所示。

表 2-4 SendKeys 用到的按键写法

按　键	代　码
BACKSPACE	{BACKSPACE}, {BS}, 或 {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} 或 {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} 或 ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} 或 {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}

续表

按　　键	代　　码
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

例如，要在一个窗口中按下退格键，书写代码的方法如下。

```
WshShell.SendKeys "{BACKSPACE}"
```

下面的实例首先启动记事本，其次输入英文单词，然后按下退格键删掉最后一个字符。

```
Public Sub Delay(Interval As Single)
    Dim timer0 As Single
    timer0 = Timer
    Do While Timer - timer0 < Interval
        DoEvents
    Loop
End Sub
Sub Test1()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Shell "notepad", vbNormalFocus
    With WS
        Delay 1
        .SendKeys "Excel VBA"
        Delay 1
        .SendKeys "{BackSpace}"
        Delay 1
        .AppActivate " 无标题 - 记事本 "
    End With
End Sub
```

代码分析：Delay 过程用于延时，Delay 1 表示程序暂停 1 秒，这个技术经常用于自动发送按键的程序中。

SendKeys "Excel VBA"，相当于依次按了 9 次键盘，每次按的都是字母键。因此向一个接受文字的区域输入英文句子，就可以采用这种方式。SendKeys "{BackSpace}" 千万不能写成 SendKeys "BackSpace"，因为这样表示连续按下多次字母键。

除了输入英文外，还可以输入键盘上的符号，例如 SendKeys "3*2=" 就自动输入一个数学题。

注意，发送一个英文字母或符号可以用花括号括起来，但是两个以上不可，例如 SendKeys "{M}" 是允许的，但 SendKeys {VBA}" 不允许。

2.8.2 多次按同一个键

如果要多次按下同一个键，键码必须放在花括号内，然后输入空格和次数。例如下面这些示例。

SendKeys "138{6 4}{8 4}" 表示输入 13866668888。

SendKeys "{Enter 3}" 表示连续按下 3 次回车键。

SendKeys "{+ 3}{—3}" 表示连续按 3 次加号，然后按 3 次减号。

利用这个特点，经常可以在文字中移动光标，例如 SendKeys "Microsoft{Left 4}{Right 2}" 表示在记事本程序中输入 Microsoft 这个单词，然后按下 4 次左箭头、2 次右箭头，从而把光标移动到 o 与 f 之间。

2.8.3 组合按键

如果要发送包含 Ctrl、Alt、Shift 的组合键，分别用 ^、%、+ 表示。

SendKeys "^o" 或者 SendKeys "^{o}" 表示按下。【 Ctrl+O 】，用来打开文件。

SendKeys "+{F3}" 表示按下快捷键【 Shift+F3 】。

SendKeys "%f" 表示按下快捷键【 Alt+F 】，经常用于显示文件菜单。

下面的代码向记事本程序输入一个英文句子，然后调出记事本程序的替换对话框快捷键【 Ctrl+H 】，把里面的字母 o 全部替换为 X。

```
Sub Test2()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Shell "notepad", vbNormalFocus
    With WS
        Delay 1
        SendKeys "Microsoft Office"
        Delay 1
        .SendKeys "^h"
        Delay 1
        .SendKeys "o"
        Delay 1
        .SendKeys "{Tab}"
        Delay 1
        .SendKeys "X"
        Delay 1
        .SendKeys "%a"
        Delay 1
        .SendKeys "{ESC}"
    End With
End Sub
```

代码分析：SendKeys "{Tab}" 表示切换控件焦点，也就是从查找文本框切换到替换文本框中。这一步是必不可少的。

在记事本程序的替换对话框中，“全部替换”按钮中有个带下画线的字母 A，意思是用【Alt+A】快捷键按下该按钮，因此采用代码 SendKeys "%a"。

最后一句代码 SendKeys "{ESC}"，表示按下【Escape】关闭替换对话框。

执行上述过程，可以看到其中的字母被替换，如图 2-47 所示。



图 2-47　使用 Sendkeys 自动操作记事本程序

2.8.4　特殊符号的输入

前面讲过，双引号中的 ^、%、+ 被转义为组合键，如果需要输出这几个字符本身，就需要把它们放在花括号内。例如，SendKeys "{+}{%}" 表示往记事本中写入 +%。

此外，要输出花括号本身，也需要把它们套在花括号内。例如 SendKeys "{}}{{}" 表示往记事本中写入 }{。

2.8.5　循环中使用按键

如果按键的内容相同或者类似，可以把 Sendkeys 方法放在循环结构中，从而减少 Sendkeys 的书写次数。

下面的代码把 Excel 单元格区域中的内容顺次打印到记事本程序中。

```
Sub Test3()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim rg As Excel.Range
    Shell "notepad", vbNormalFocus
    With WS
        For Each rg In Sheet2.Range("A1:C4")
            Delay 1
            .SendKeys rg.Value
            .SendKeys "{enter}"
        Next rg
    End With
End Sub
```

可以看出代码中只有两处 Sendkeys，就把 12 个单元格的内容发送到记事本程序中，如图 2-48 所示。

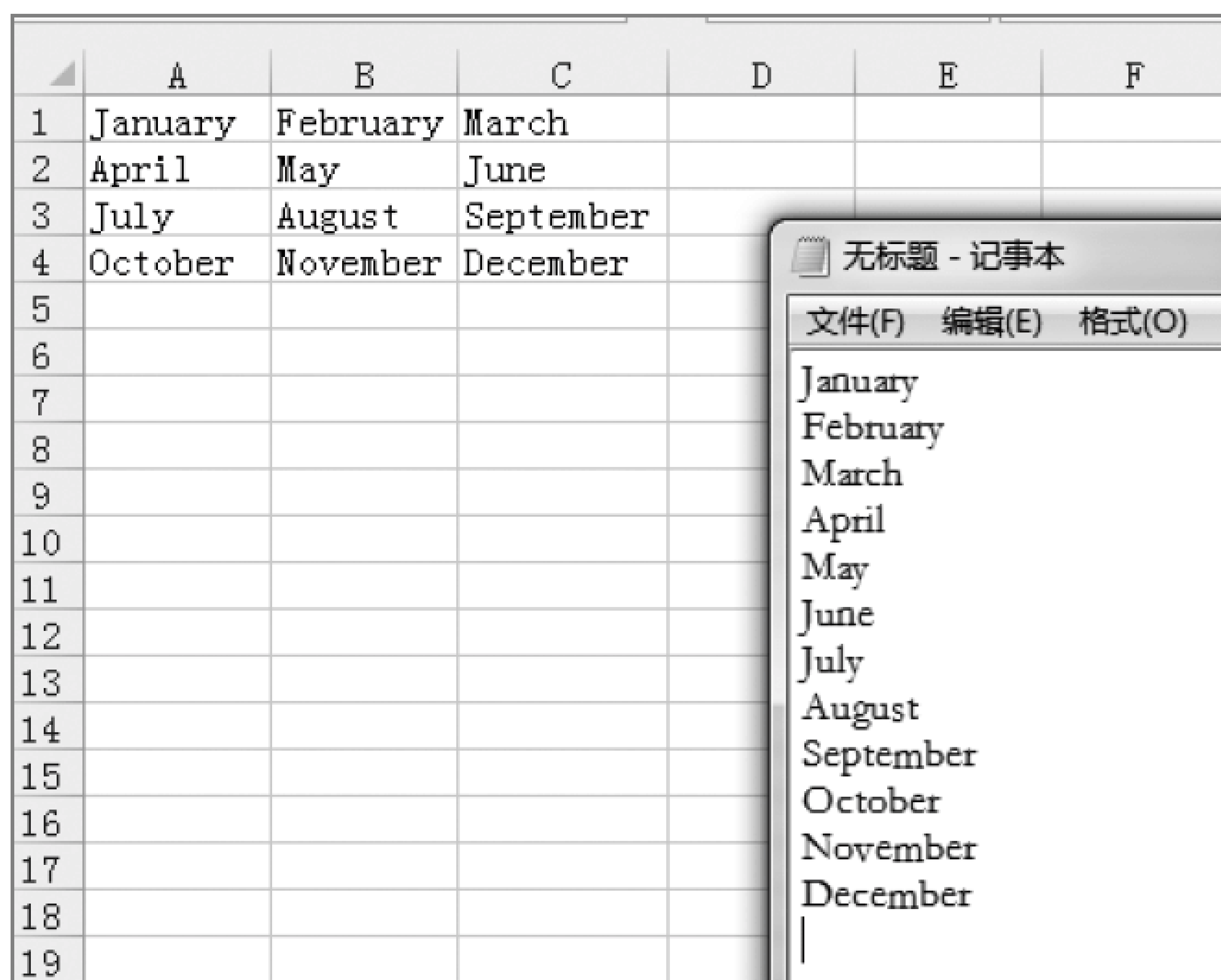


图 2-48　循环使用 Sendkeys

下面是一个技术点比较综合的实例，希望读者仔细分析其细节。

```
Sub AutoRun()
    Dim WS As New IWshRuntimeLibrary.WshShell
    Dim Style As IWshRuntimeLibrary.WshWindowStyle
    Dim Result As Long
    With WS
        Delay 3
        Style = WshNormalFocus
        .Run Command:="notepad.exe", WindowStyle:=Style, WaitOnReturn:=False
        Delay 1
        .SendKeys "{* 5}By Liu YongFu{* 5}"            '在记事本程序中打字
        Delay 1
        .SendKeys "{Enter 2}"                          '按 2 次换行键
        Delay 1
        .SendKeys "{{}I will enlarge font-size!{}}"    '在记事本程序中打字
        Delay 1
        .SendKeys "{Enter 2}"                          '按 2 次换行键
        Delay 1
        .SendKeys "%o"                   '单击记事本程序的"格式"菜单
        Delay 1
        .SendKeys "f"                                  '单击"字体"
        Delay 1
        .SendKeys "{TAB 2}"              '连续按 3 下 Tab 键，焦点切换到字号组合框
        Delay 1
        .SendKeys "{UP 2}"               '连续按 2 次上箭头，增大字号
        Delay 1
        .SendKeys "{Enter}"                            '确认并关闭字体对话框
        Delay 1
        .SendKeys "Now,Exit Notepad!"                  '继续打字
        Delay 1
        .SendKeys "%{f}x"                '连续按下快捷键【Alt+F】、X 键
```

```
        Delay 1
        .SendKeys "n"                    '询问是否保存，选择 "No"
    End With
End Sub
```

2.8.6　关于自动按键的补充说明

也就是说，不仅可以用 Sendkeys 方法输入文本，还可以自动单击窗口的菜单、设定对话框中的控件参数等。

但是 Sendkeys 存在如下几个不足。

- 不能发送中文字符。
- 不能发送一部分特殊按键，例如 Windows 键（Ctrl 和 Alt 之间的键）。
- 被操作的窗口不能被遮挡，不能失去焦点。

对于中文或其他字符串的发送，可以借助剪贴板的功能，先把中文发送到系统的剪贴板上，然后发送按键【Ctrl+V】再粘贴中文即可。

如果要按下特殊按键，还需要借助 API 函数，本书暂不讲解 API 函数用法。

2.9　使用 WshNetwork 对象

IWshRuntimeLibrary 下面的 WshNetwork 对象可以操作局域网多台计算机的对象，其重要属性如下。

- ComputerName：返回计算机名称。
- UserName：返回用户名。

重要方法如下。

- MapNetworkDrive：映射网络驱动器。
- RemoveNetworkDrive：移除指定的网络驱动器。

与打印机有关的成员如下。

- EnumPrinterConnections：枚举所有打印机。
- SetDefaultPrinter：设置默认打印机。

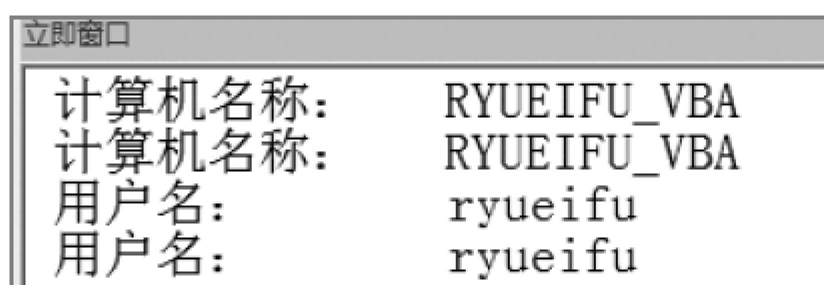
2.9.1　返回计算机属性

下面的程序返回当前计算机的名称和用户名。

```
Sub GetComputerProperty()
    Dim network As New IWshRuntimeLibrary.WshNetwork
    With network
        Debug.Print "计算机名称：", .ComputerName
        Debug.Print "计算机名称：", Environ("ComputerName")
        Debug.Print "用户名：", .UserName
        Debug.Print "用户名：", Environ("UserName")
```

```
    End With
End Sub
```

运行上述程序，立即窗口的打印结果如图 2-49 所示。


立即窗口

```
计算机名称:    RYUEIFU_VBA
计算机名称:    RYUEIFU_VBA
用户名:        ryueifu
用户名:        ryueifu
```

图 2-49 查看计算机名和用户名

2.9.2 映射网络驱动器

映射网络驱动器功能可以把网络中的其他计算机或服务器中的磁盘、路径映射为当前计算机中的一个分区。WshNetWork 下面的 MapNetworkDrive 方法可以实现这一功能，其参数如下。

- ❑ LocalName：本地驱动器名称。
- ❑ RemoteName：远程服务器或计算机中的路径，一般以 \\ 加上计算机名称或 IP 地址开头。
- ❑ UpdateProfile：是否保存映射信息到本地计算机中，默认值为 False。
- ❑ UserName：远程服务器或计算机的用户名。
- ❑ Password：远程服务器或计算机的密码。

假设有一台远程计算机的名称为 ryueifu_VBA，有如下路径：

```
D:\TEXTBOOK\Python
```

运行如下程序就可以把该路径映射为本地计算机的 Z 分区。

```
Sub AllocateDrive()
    Dim network As New IWshRuntimeLibrary.WshNetwork
    network.MapNetworkDrive LocalName:="Z:", RemoteName:="\\ryueifu_VBA\D$\
TEXTBOOK\Python", UpdateProfile:=False
End Sub
```

运行上述程序，本地计算机的资源管理器中多出了 Z 分区，如图 2-50 所示。



图 2-50 自动映射网络驱动器

与之相反，使用 RemoveNetworkDrive 可以移除指定的映射分区。

```
Sub RemoveDrive()
    Dim network As New IWshRuntimeLibrary.WshNetwork
    network.RemoveNetworkDrive Name:="Z:", Force:=True, UpdateProfile:=False
```

```
End Sub
```

运行上述程序，Z 分区自动消失。

2.9.3　操作打印机

WshNetwork 对象有很多用于操作打印机的成员，例如 EnumPrinterConnections 可以用于枚举计算机中所有的打印机端口和名称。

下面的程序枚举当前计算机上所有的打印机名称。

```
Sub LoopPrinters()
    Dim network As New IWshRuntimeLibrary.WshNetwork
    Dim i As Integer
    Dim Printers As IWshRuntimeLibrary.WshCollection
    Set Printers = network.EnumPrinterConnections
    For i = 1 To Printers.Count Step 2
        Debug.Print Printers(i) '打印机名称
    Next i
End Sub
```

代码分析：如果把 Debug.Print Printers(i) 中的 i 改成 i—1，则打印出来的是每个打印机的端口名称。

运行上述程序，立即窗口打印出所有打印机名称，如图 2-51 所示。

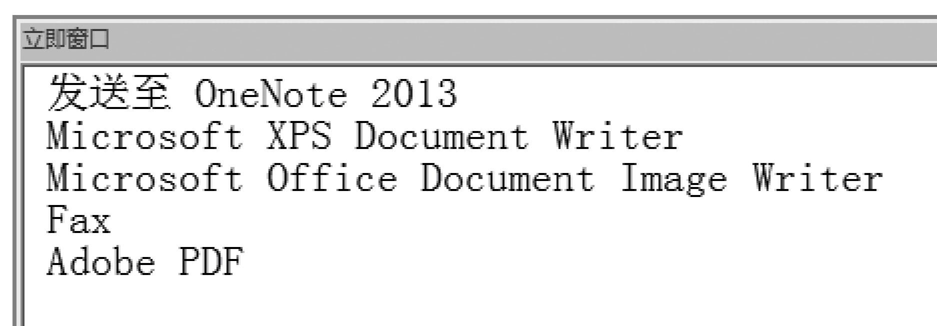


图 2-51　遍历所有打印机

SetDefaultPrinter 方法则可以设置默认打印机。

```
Sub 设置默认打印机()
    Dim network As New IWshRuntimeLibrary.WshNetwork
    network.SetDefaultPrinter Name:="Adobe PDF"
    Debug.Print "默认打印机: ", Application.ActivePrinter
End Sub
```

运行以上程序，更改默认打印机，然后打印出默认打印机的名称，如图 2-52 所示。



图 2-52　自动设置默认打印机

以上内容的源代码文件为“实例文档 09.xlsm”。

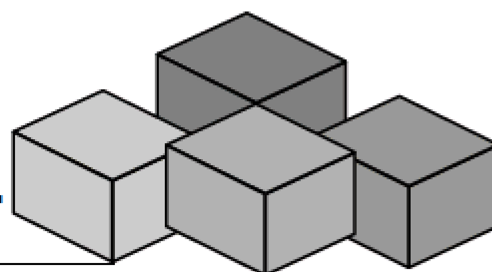
2.10 本章小结

本章讲解了 Shell 函数调用其他可执行文件的方法，一定要注意 Shell 函数的异步问题。

在注册表读写方面，VBA 内置函数 GetSetting、SaveSetting 等只能操作注册表中的一小部分，而使用 WshShell 对象的 RegRead、RegWrite、RegDelete 方法可以读写注册表的任何位置。

第 3 章

处理压缩文件



在日常办公中，经常用到文件的压缩与解压缩，这些操作也可以用 VBA 实现自动化。另外，Office 2007 以上版本创建的文档其实也是压缩包文件，只不过扩展名看起来是 Office 的扩展名，因此，学习压缩和解压缩的知识，有助于理解 Office 开发中自定义 Office 界面方面的知识。

文件的压缩和解压缩操作有以下两个主要方式。

- ❑ Shell 函数调用电脑默认的压缩工具。
- ❑ 使用 Shell32 对象操作 .zip 压缩文件。

第一种方式通用性比较强，几乎可以操作任意扩展名的压缩包，缺点是计算机必须安装了压缩软件。

针对第二种方式，不需要安装压缩软件，用代码即可实现压缩和解压缩，但只限于扩展名为 .zip 的压缩包。

本章包括用 Shell 函数调用 WinRAR 压缩软件以及使用 Shell32 对象操作 .zip 压缩文件两大部分内容。

本章用到的外部引用和重要对象如下。

- ❑ Microsoft Shell Controls And Automation
  - ➢ Shell32.Shell

3.1 Shell 调用 WinRAR

WinRAR 是一个文件压缩管理共享软件，由 Eugene Roshal（所以 RAR 的全名是 Roshal ARchive）开发。首个公开版本 RAR 1.3 发布于 1993 年。

WinRAR 可以把文件（夹）压缩为 .rar 或 .zip 格式，如图 3-1 所示。

WinRAR 可以解压的格式有：.CAB、.ARJ、.LZH、.TAR、.GZ、.ACE、.UUE、.BZ2、.JAR、.ISO、.Z、.7Z、.RAR5。

启动 WinRAR 软件，单击 WinRAR 软件的菜单【选项 / 设置】，弹出“设置”对话框，切换到“集成”选项卡，可以设置 WinRAR 能够解压的文件格式，如图 3-2 所示。



图 3-1 “压缩文件名和参数”对话框

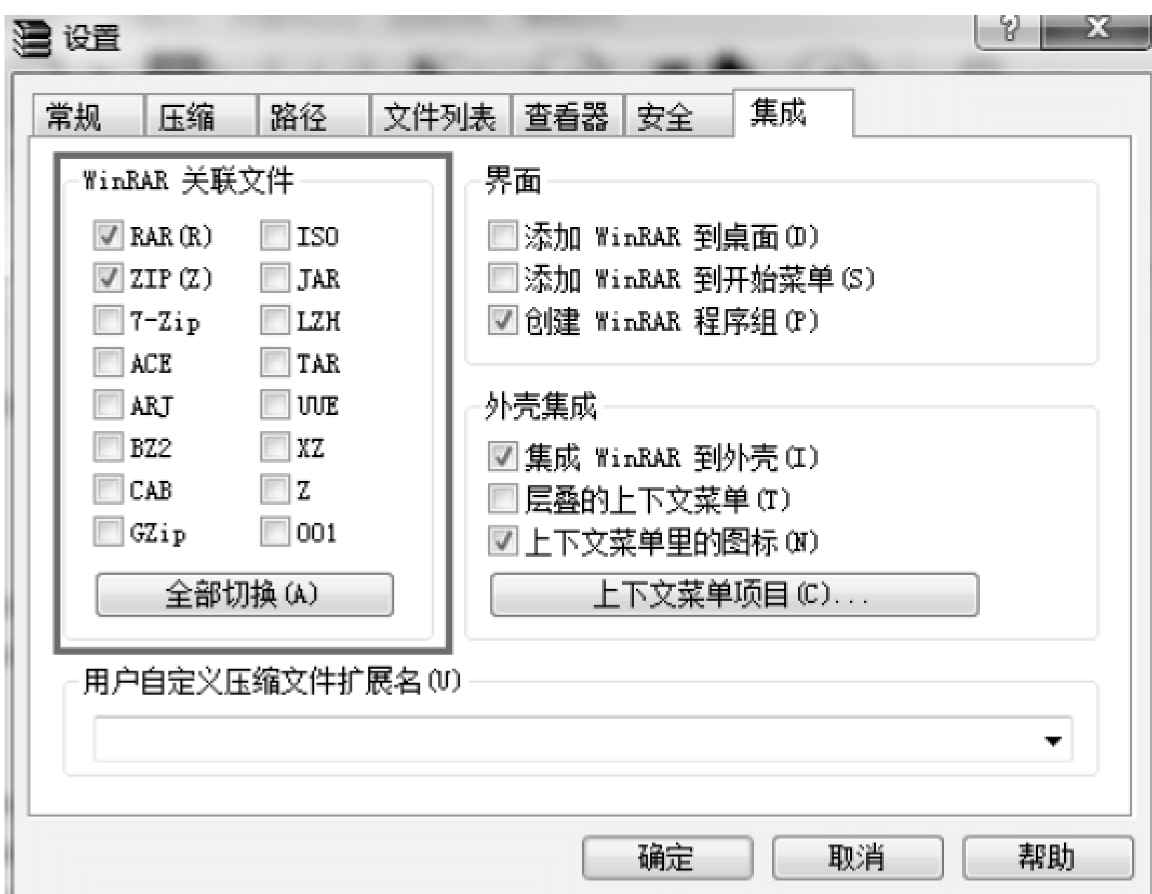


图 3-2 勾选关联的扩展名

3.1.1 获取 WinRAR 可执行文件路径

WinRAR 的执行文件一般情况下位于 C:\Program Files\WinRAR\WinRAR.exe，如果个别计算机把这个软件安装到其他位置，使用前面讲过的 WshShell 对象的 RegRead 方法读取注册表可以获取其路径。

下面的 GetSetupPath 函数用来获取指定程序名的安装路径。

```
Public Function GetSetupPath(AppName As String)
    Dim WSH As Object
    Set WSH = CreateObject("WScript.Shell")
    GetSetupPath = WSH.RegRead("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\App Paths\" & AppName & "\Path")
    Set WSH = Nothing
End Function
```

运行下面的过程，可以获取 WinRAR 软件的安装路径以及 PowerPoint 的安装路径。

```
Sub Test()
    Debug.Print GetSetupPath("WinRAR.exe")
    Debug.Print GetSetupPath("Powerpnt.exe")
End Sub
```

上述程序的运行结果如图 3-3 所示。

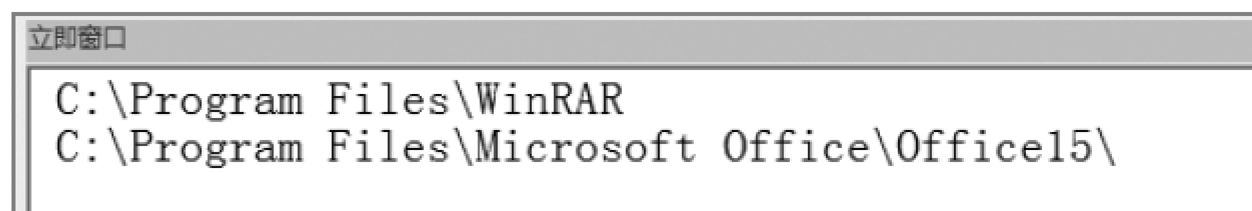


图 3-3 从注册表中获取应用程序的所在路径

3.1.2　命令和开关

获取到 WinRAR.exe 的所在路径，就可以使用 Shell 函数调用这个可执行文件完成压缩和解压缩操作。

调用格式如下。

```
Shell "WinRAR.exe 的路径 命令 开关 压缩包路径 文件路径",vbNormalFocus(或者 vbHide)
```

下面通过一个实例来介绍一下各参数的构造方法。

```
Shell "C:\Program Files\WinRAR\WinRAR.exe A C:\temp\Regdll.rar C:\temp\65.
png", vbNormalFocus
```

以上语句的功能是，调用 WinRAR.exe 把 65.png 图片文件压缩到 Regdll.rar 这个压缩包中。可以看出各个参数之间用空格隔开，全部放入双引号内，形成了一个长的字符串。其中的 A 就是一个命令，表示压缩，上面这个实例没有用到开关参数。

下面分别介绍一下 WinRAR 的命令参数和开关参数。命令参数的功能是告知 WinRAR 要执行什么操作，是压缩、解压缩还是删除。开关参数是对命令参数的补充说明。

WinRAR 命令参数

单击 WinRAR 的菜单【帮助 / 帮助主题】，可以打开其帮助文件，依次展开节点“命令行模式 / 命令行”，可以看到所有命令参数的说明，如图 3-4 所示。



图 3-4　WinRAR 的命令参数帮助

最常用的 4 个命令参数及其功能如下。

- A：压缩，添加到压缩文件中。
- D：删除，从压缩包中删除文件。
- E：解压缩到当前目录。
- X：以完全路径解压。

可以看出，从压缩包中解压出内容，有 E 和 X 两个命令参数。其实，E 命令参数等价于 WinRAR 解压参数中的“不要提取路径”；X 命令参数等价于“提取完整路径”，使用 WinRAR 软件解压一个压缩包时，在“高级”选项卡里可以看到解压方式选项，如图 3-5 所示。



图 3-5　命令参数相应的含义

简言之，E 就是忽略压缩包中的路径，释放所有文件到目标文件夹，而 X 则按照压缩包原有的路径结构释放到目标文件夹。

WinRAR 开关参数

在 WinRAR 软件的帮助文件中，依次展开节点“命令行模式 / 参数”，可以看到所有开关参数的说明，如图 3-6 所示。

下面是比较常用的开关参数。

- -ep：忽略路径。
- -ep1：忽略基准路径，但保持现有文件层次结构。
- -p 或 -hp：压缩时加密码。
- -df：压缩后删除原文件。
- -dr：压缩后删除原文件到回收站。

大致了解命令参数和开关参数后，下面通过具体实例加深学习。

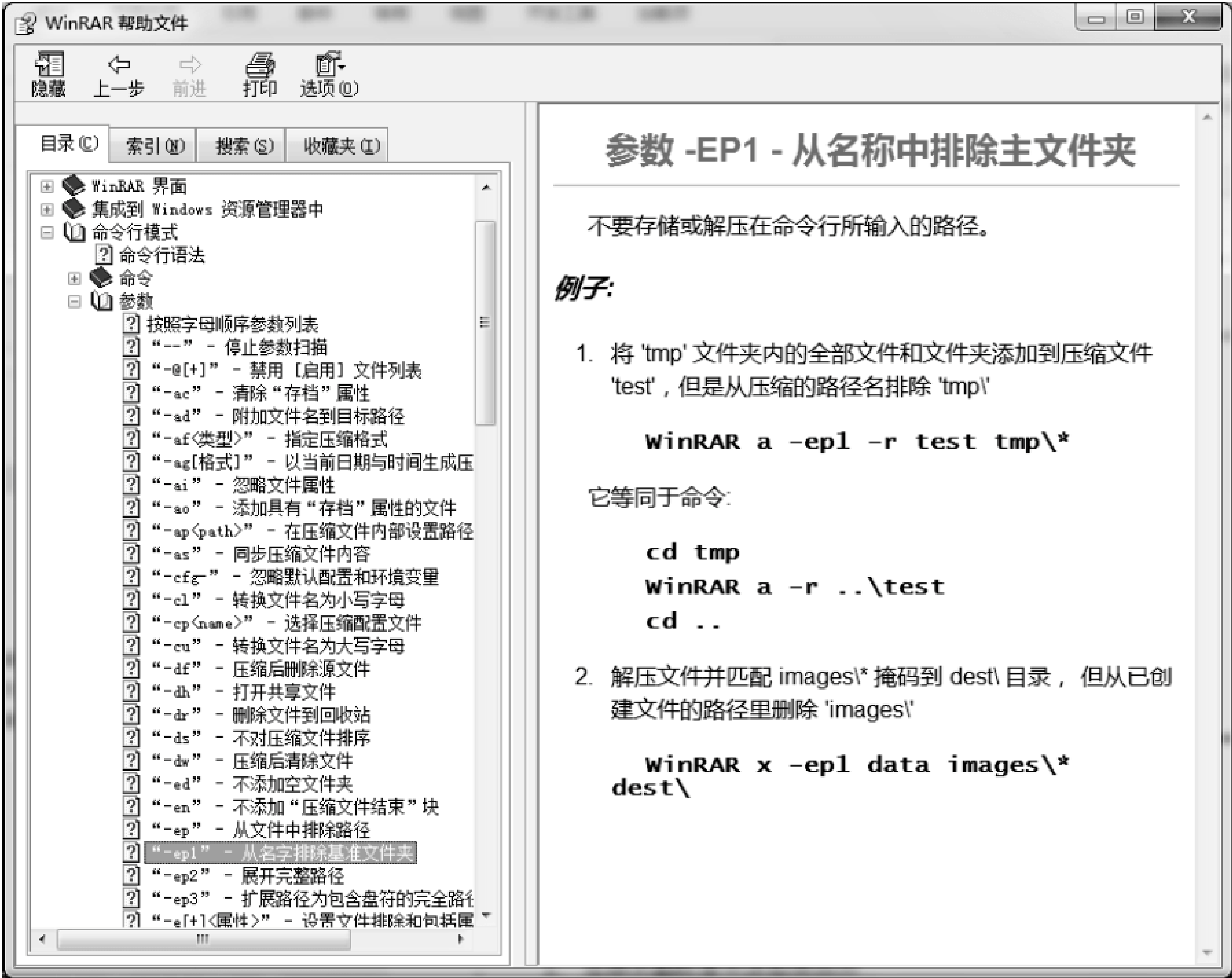


图 3-6　WinRAR 开关参数帮助

3.1.3　压缩

假设文件夹“东北三省”中的内容如图 3-7 所示。

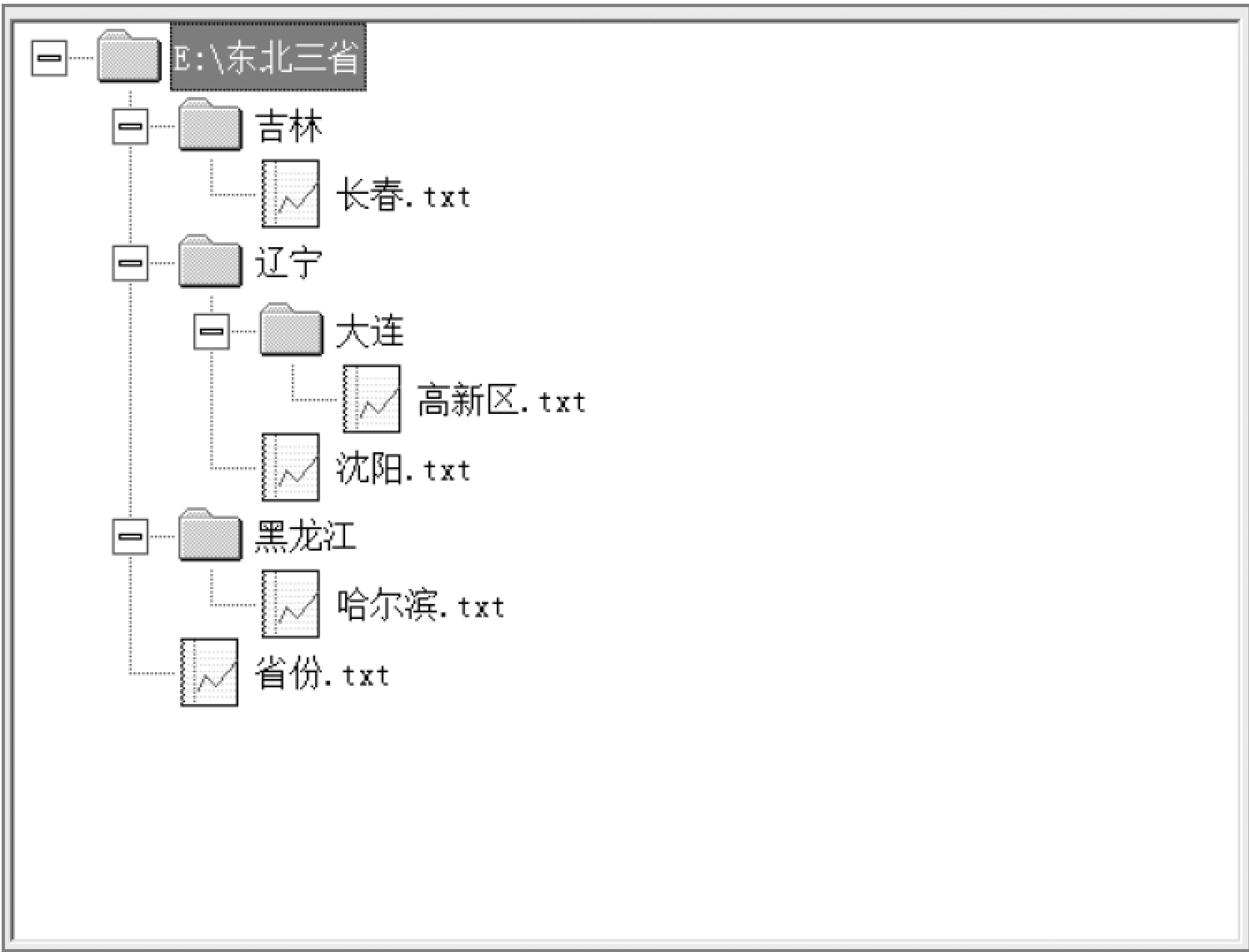


图 3-7　文件夹内容的示意图

如果采用命令 A-ep 则是忽略所有路径，也就是忽略文件夹及其子文件夹，把“东北三省”下面管辖的所有文件（含递归）压缩进去。完整代码如下。

```
    Public Function GetSetupPath(AppName As String)
        Dim WS As New IWshRuntimeLibrary.WshShell  '前期绑定
        Set WS = CreateObject("WScript.Shell")
        GetSetupPath = WS.RegRead("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\App Paths\" & AppName & "\Path")
        Set WS = Nothing
    End Function
    Private Function AddQuote(path As String) As String
        AddQuote = Chr(34) & path & Chr(34) & " "
    End Function

    Sub 文件夹及其内容添加到压缩包()
        Const WorkDir As String = "E:\"      '默认目录
        Dim WinrarExe As String              'WinRAR 可执行文件路径
        Dim Command As String                '命令、开关参数
        Dim RarFile As String                '压缩包路径
        Dim Contents As String               '文件、文件夹的路径
        WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
        Command = " A -ep "
        RarFile = WorkDir & "package1.rar"
        Contents = WorkDir & "东北三省"      '把"东北三省"文件夹及其内容添加到压缩包
        Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote (Contents),
vbNormalFocus
    End Sub
```

代码分析：函数 GetSetupPath 用来获取 WinRAR 软件的安装目录，函数 AddQuote 用来处理 Shell 命令路径中的空格。要注意 Command 中要保留必要的空格。

以上代码段中，最重要的一句就是最后 Shell 函数的应用。

运行上面的“文件夹及其内容添加到压缩包”过程，会在 E: 盘下生成 package1.rar 压缩包，手工打开后，如图 3-8 所示。

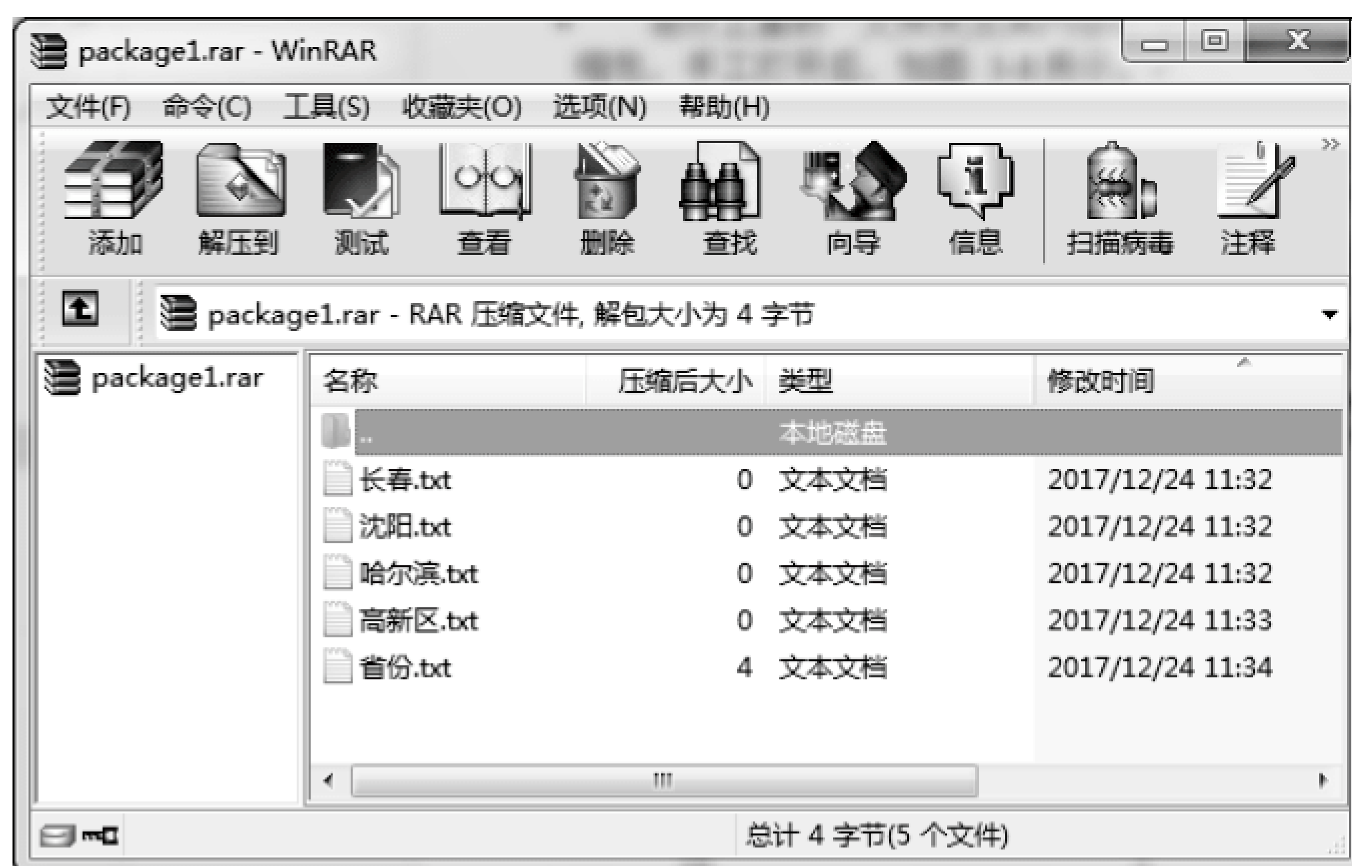


图 3-8　自动执行压缩

可以看到，A-ep 命令把文件夹中所有的“文件”掏出来，放入压缩包，而不管这些文件原先在何处。

现在只需把上述代码中的 Command 换成 "A-ep1"，删除原先的压缩包，再运行一次程序，效果如图 3-9 所示。



图 3-9　连文件夹一起压缩

可以看出，该命令保留了原先文件结构。因此，可以简单地理解为 -ep 参数只压缩文件，-ep1 带文件夹压缩。

在实际应用中，根据需要选择开关参数即可。

3.1.4　解压缩

解压缩是将压缩包中的内容释放到磁盘下的操作。解压缩的命令有 E 和 X。

下面的过程把压缩包 package1.rar 中所有的文件解压到 Destination 文件夹下。

```
Sub 解压 ()
    Const WorkDir As String = "E:\"      ' 默认目录
    Dim WinrarExe As String              'WinRAR 可执行文件路径
    Dim Command As String                ' 命令、开关参数
    Dim RarFile As String                ' 压缩包路径
    Dim Contents As String               ' 文件、文件夹的路径
    WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
    Command = " E "
    RarFile = WorkDir & "package1.rar"
    Contents = WorkDir & "Destination"  ' 释放到 Destination 路径下
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote (Contents),
vbNormalFocus
End Sub
```

运行上述程序，把压缩包中的文件直接释放到目标文件夹，如图 3-10 所示。

如果把 Command 改为 Command = " X "，再次运行上述程序，压缩包中的文件夹和文件一律解压到目标文件夹中，如图 3-11 所示。

上面的实例把压缩包中所有内容解压到目标文件夹，使用以下代码可以解压压缩包中指定路径的文件，而不是解压全部文件。

```
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & " 东北三省\辽宁\*.* "
& AddQuote(Contents), vbNormalFocus
```

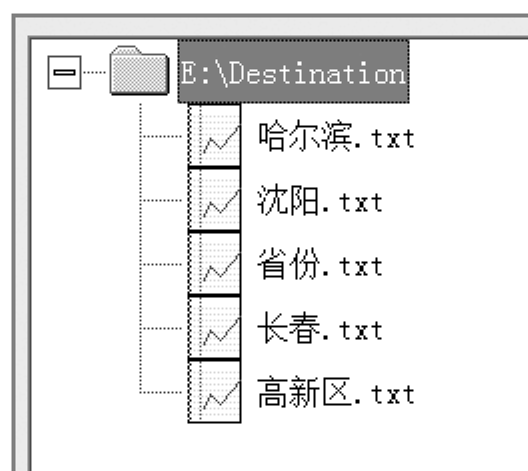


图 3-10 自动解压

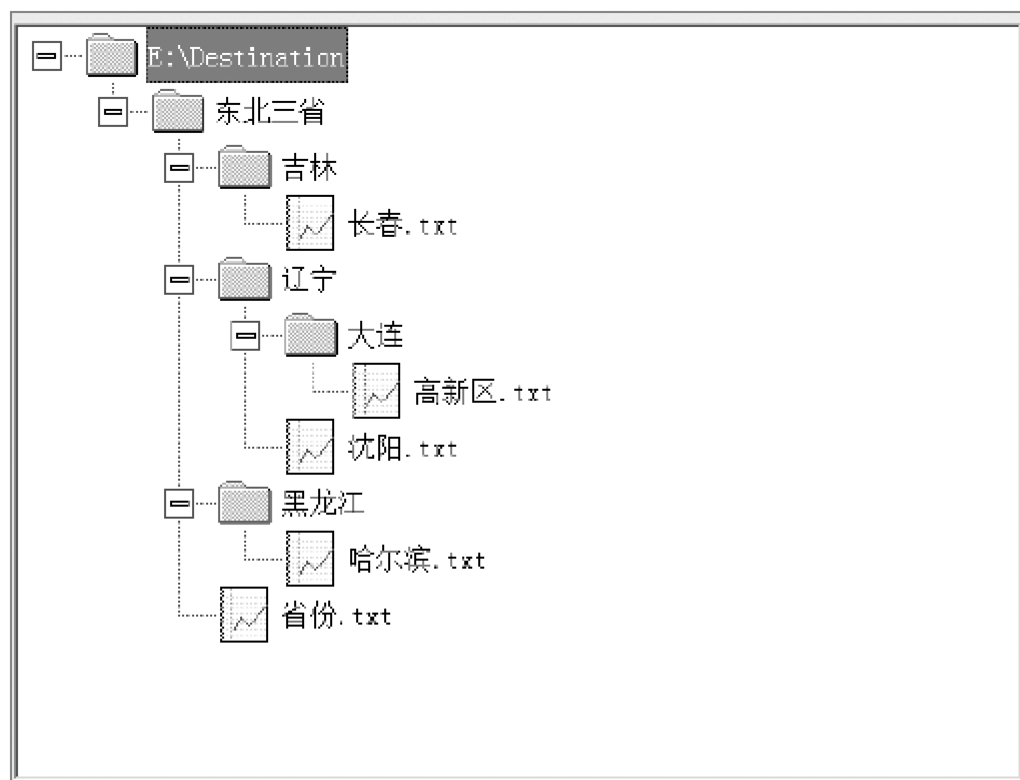


图 3-11 连文件夹一起解压

3.1.5 删除

WinRAR 使用命令 D 删除压缩包中的文件或路径。

下面的实例把 package1.rar 压缩包中的“吉林”文件夹删除。

```
Sub 删除压缩包中内容()
    Const WorkDir As String = "E:\"             '默认目录
    Dim WinrarExe As String                     'WinRAR 可执行文件路径
    Dim Command As String                       '命令、开关参数
    Dim RarFile As String                       '压缩包路径
    Dim Contents As String                      '压缩包中待删除的文件、文件夹的路径
    WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
    Command = " D "
    RarFile = WorkDir & "package1.rar"
    Contents = "东北三省\吉林"   '把"东北三省\吉林"压缩包中的路径删除
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote (Contents),
vbNormalFocus
End Sub
```

运行上述程序，删除压缩包中的“吉林”文件夹，如图 3-12 所示。



图 3-12 删除压缩包中指定的文件夹

3.1.6 使用通配符

无论是压缩、解压缩，还是删除命令，路径设置中均可使用通配符。* 表示 0 个以上任

意字符，? 表示 1 个任意字符。

下面的程序把文件夹下所有 4 位扩展名的 Word 文档添加到 Word.rar 压缩包中。

```
Sub 指定类型的文件添加到压缩包 ()
    Const WorkDir As String = "C:\temp\"        '默认目录
    Dim WinrarExe As String                    'WinRAR 可执行文件路径
    Dim Command As String                      '命令、开关参数
    Dim RarFile As String                      '压缩包路径
    Dim Contents As String                     '文件、文件夹的路径
    WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
    Command = " A -ep "
    RarFile = WorkDir & "Word.rar"
    Contents = WorkDir & "*.doc?" '把四位扩展名的 Word 文档 (docx、docm) 添加到压缩包。
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote (Contents),
vbNormalFocus
End Sub
```

运行上述程序，将文件夹中所有的 Word 文档添加到压缩包，如图 3-13 所示。

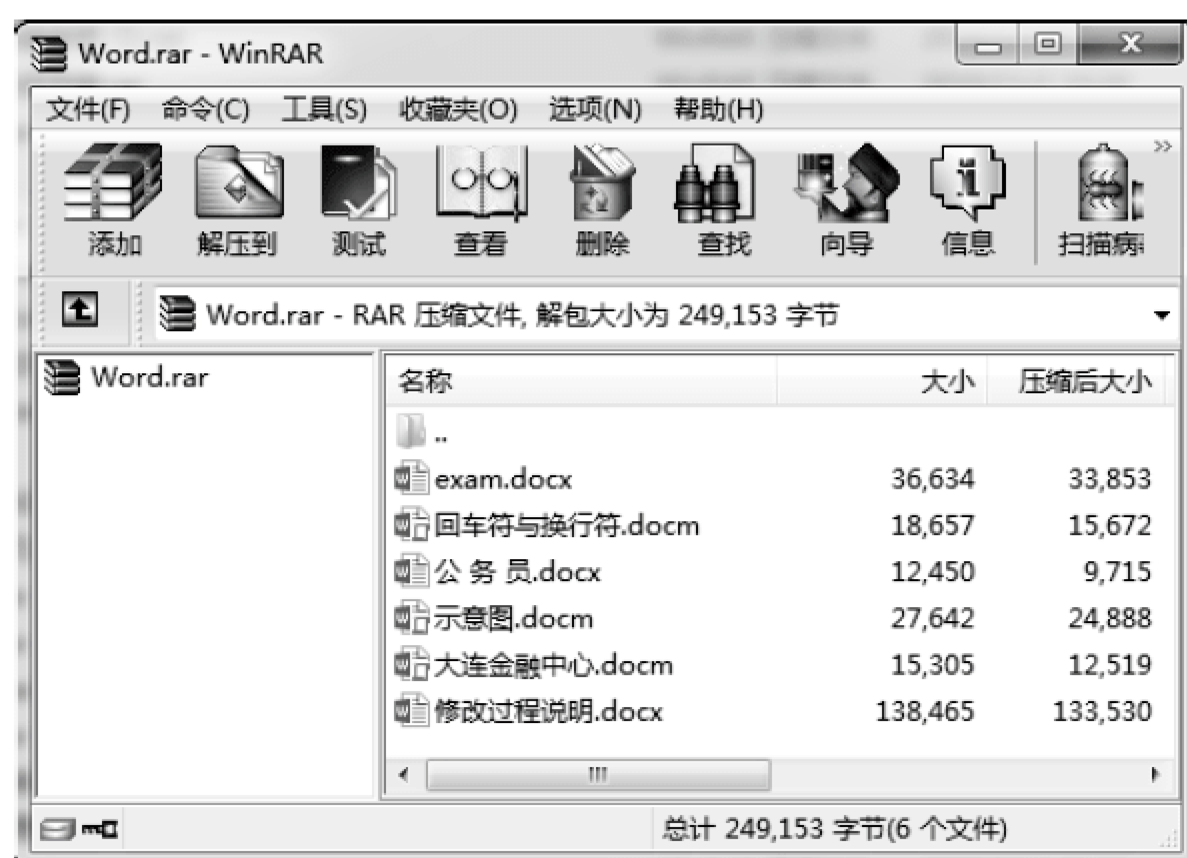


图 3-13　批量压缩特定类型的文件

下面的代码从 Word.rar 压缩包中删除启用宏的 Word 文档（扩展名为 .docm）。

```
Sub 删除压缩包指定类型的文件 ()
    Const WorkDir As String = "C:\temp\"    '默认目录
    Dim WinrarExe As String                'WinRAR 可执行文件路径
    Dim Command As String                  '命令、开关参数
    Dim RarFile As String                  '压缩包路径
    Dim Contents As String                 '文件、文件夹的路径
    WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
    Command = " D "
    RarFile = WorkDir & "Word.rar"
    Contents = "*.docm"                    '删除扩展名为 docm 的文件
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote(Contents),
vbNormalFocus
End Sub
```

需要注意的是，由于是从压缩包里面删除内容，所以代码中 Content 的赋值不需要 WorkDir。

运行上述程序，删除压缩包中扩展名为 docm 的文件，如图 3-14 所示。



图 3-14 从压缩包中删除指定扩展名的文件

3.1.7 处理压缩包的密码

使用 -p 或 -hp 开关参数，可以压缩为加密的文件，也可以从添加了密码的压缩包中解压文件。-p 后面带上密码，表示普通加密，手工双击压缩包，会看到压缩包中的文件列表，每个文件后面有 *。

下面的程序把文件夹中所有扩展名为 .pdf 的文件添加到压缩包，并且设置解压密码为 ryueifu。

```
    Sub 加密的压缩包 ()
        Const WorkDir As String = "C:\temp\"        ' 默认目录
        Dim WinrarExe As String                     'WinRAR 可执行文件路径
        Dim Command As String                       ' 命令、开关参数
        Dim RarFile As String                       ' 压缩包路径
        Dim Contents As String                      ' 文件、文件夹的路径
        WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
        Command = " A -ep -pryueifu "
        RarFile = WorkDir & "Lock.rar"
        Contents = WorkDir & "*.pdf"
        Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote(Contents),
vbNormalFocus
    End Sub
```

运行上述程序，然后打开压缩包，会看到处于加密状态，如图 3-15 所示。



图 3-15 自动压缩并设置解压密码

如果使用开关参数 -hp，表示高度加密，这种方式生成的压缩包，连其中的文件列表也看不到，如图 3-16 所示。

对于设置了密码的压缩包，可以在解压命令中把密码传递进去。

下面的程序把刚刚加密生成的 Lock.rar 解压到 test 文件夹中。



图 3-16 使用 -hp 参数高度加密

```
Sub 解压带密码的压缩包 ()
    Const WorkDir As String = "C:\temp\"
                                                    ' 默认目录
    Dim WinrarExe As String                         'WinRAR 可执行文件路径
    Dim Command As String                           ' 命令、开关参数
    Dim RarFile As String                           ' 压缩包路径
    Dim Contents As String                          ' 文件、文件夹的路径
    WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
    Command = " X -pryueifu "
    RarFile = WorkDir & "Lock.rar"
    Contents = WorkDir & "test"                     ' 解压到 test 文件夹
    Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & AddQuote (Contents),
vbNormalFocus
End Sub
```

3.1.8 使用 WinRAR 修改 Office 文档

Office 2007 以上版本的 Office 文档（.docx、.xlsx、.pptx 等格式）其实是一种压缩包格式，使用 WinRAR 可以直接打开。下面介绍一下用 WinRAR 查看和修改 Excel 文件的方法。

实例文件“example01.xlsx”有 3 个工作表，表名从左到右依次分别为 Jan、Feb、Mar，如图 3-17 所示。

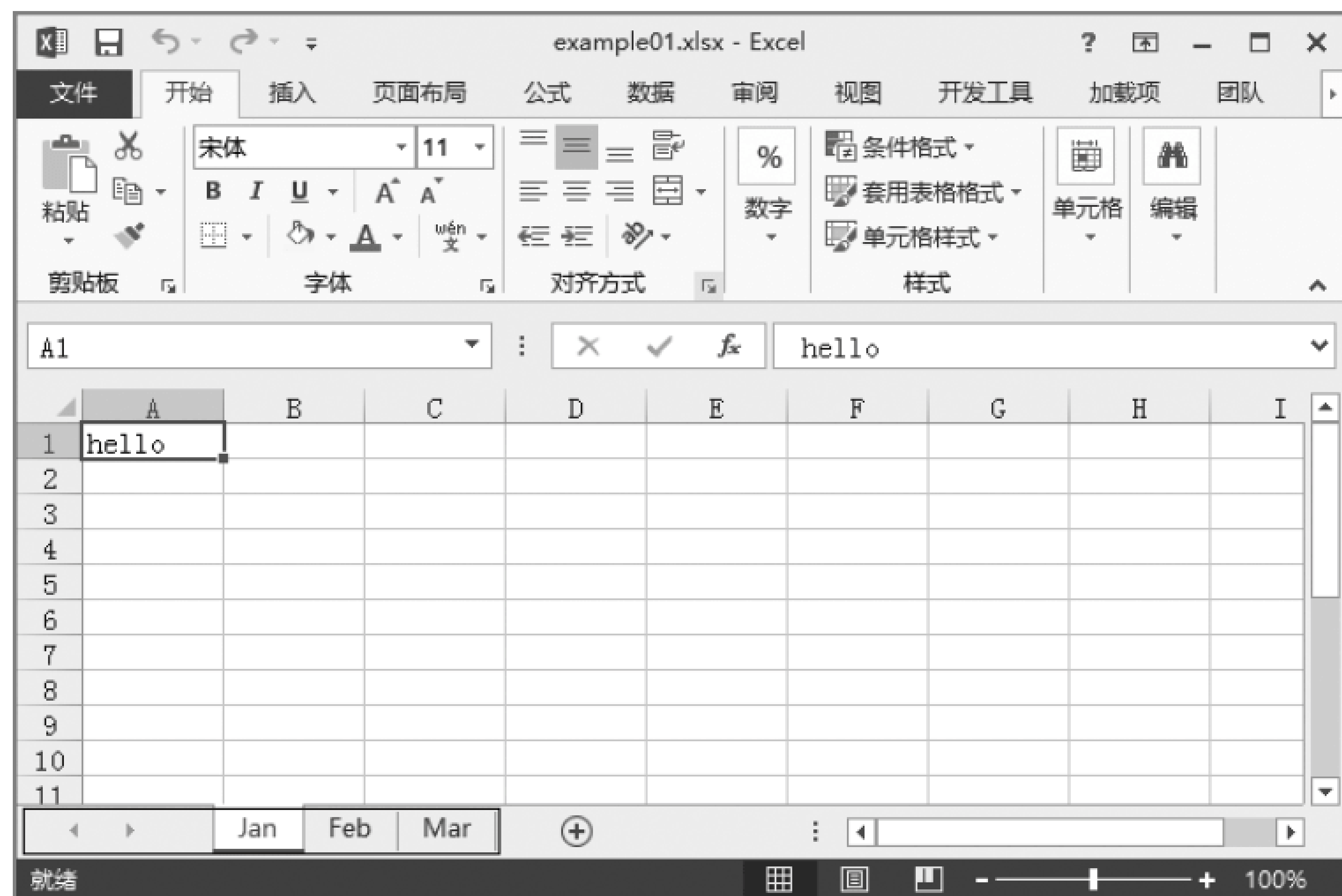


图 3-17 Excel 工作簿文件

在 Excel 中关闭该文件，然后打开 WinRAR 软件，按下快捷键【Ctrl+O】，浏览到 example01.xlsx，如图 3-18 所示。

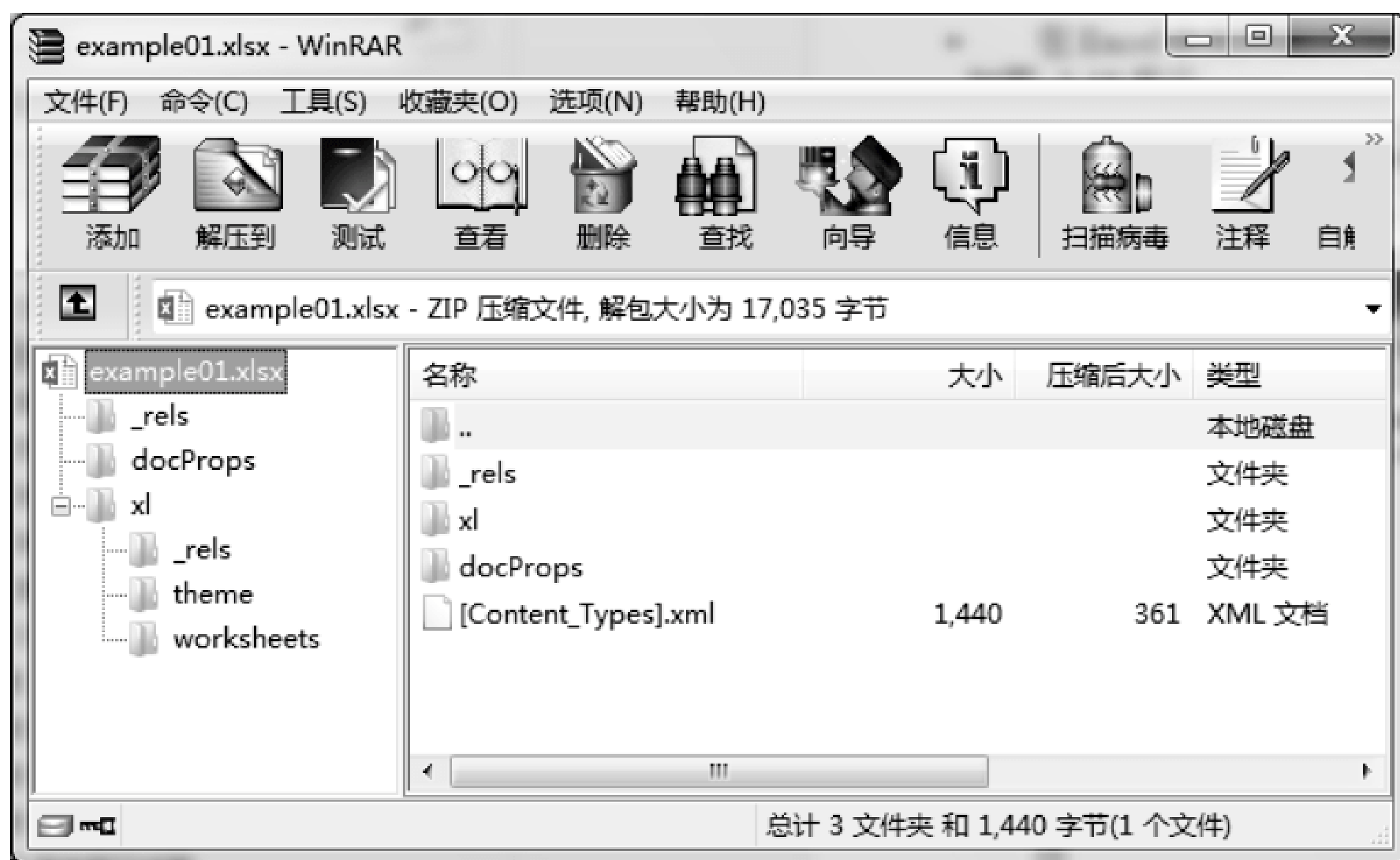


图 3-18 使用 WinRAR 打开 Excel 文件

在 WinRAR 中看到 Excel 文件由 3 个文件夹和一个文件构成，继续展开名为 xl 的文件夹，可以看到和工作表信息有关的内容，如图 3-19 所示。

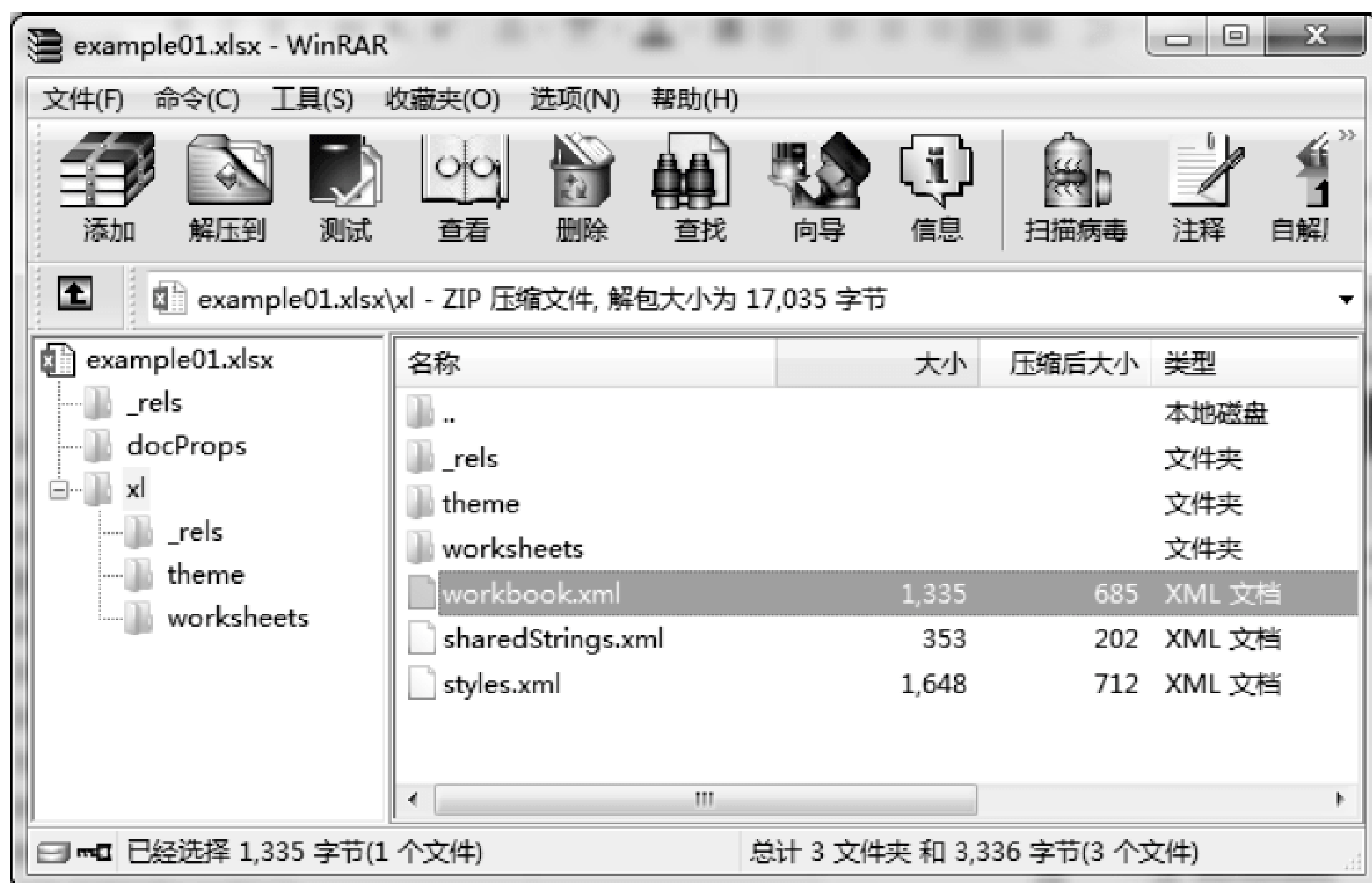


图 3-19 查看 Excel文件的内容

手工把 example01.xlsx 解压到磁盘下，生成了一些扩展名为 .xml 的文件。双击 Workbook.xml，会在 IE 浏览器中打开该文件，如图 3-20 所示。

其中 <Sheets> 这个节点中存储的就是各个工作表的名称和顺序。

xml 文件可以用记事本程序编辑，因此接下来用记事本程序打开 Workbook.xml 文件，把 Jan 这个工作表移动到最后，并且把 Feb 这个表的名称修改为“二月”。修改好后，在 IE 中再次预览，如图 3-21 所示。

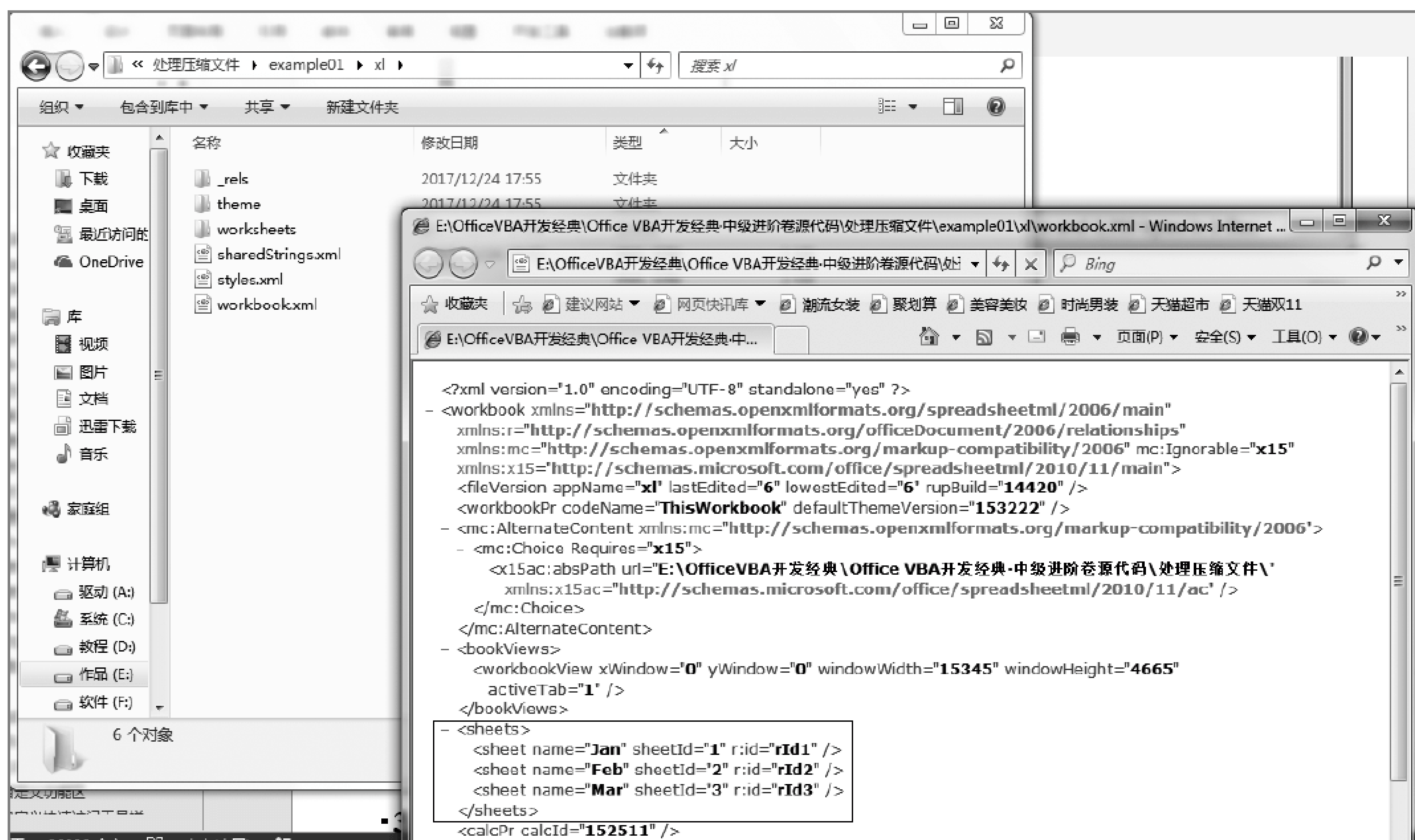


图 3-20　查看 Excel 文件内部的部署清单

```
- <sheets>
    <sheet name="二月" sheetId="2" r:id="rId2" />
    <sheet name="Mar" sheetId="3" r:id="rId3" />
    <sheet name="Jan" sheetId="1" r:id="rId1" />
  </sheets>
```

图 3-21　调整工作表的 XML 代码

然后把修改了的 Workbook.xml 文件压入 example01.xlsx 工作簿中，在 Excel 中再次打开，看到工作表的名称和次序发生了变化，如图 3-22 所示。

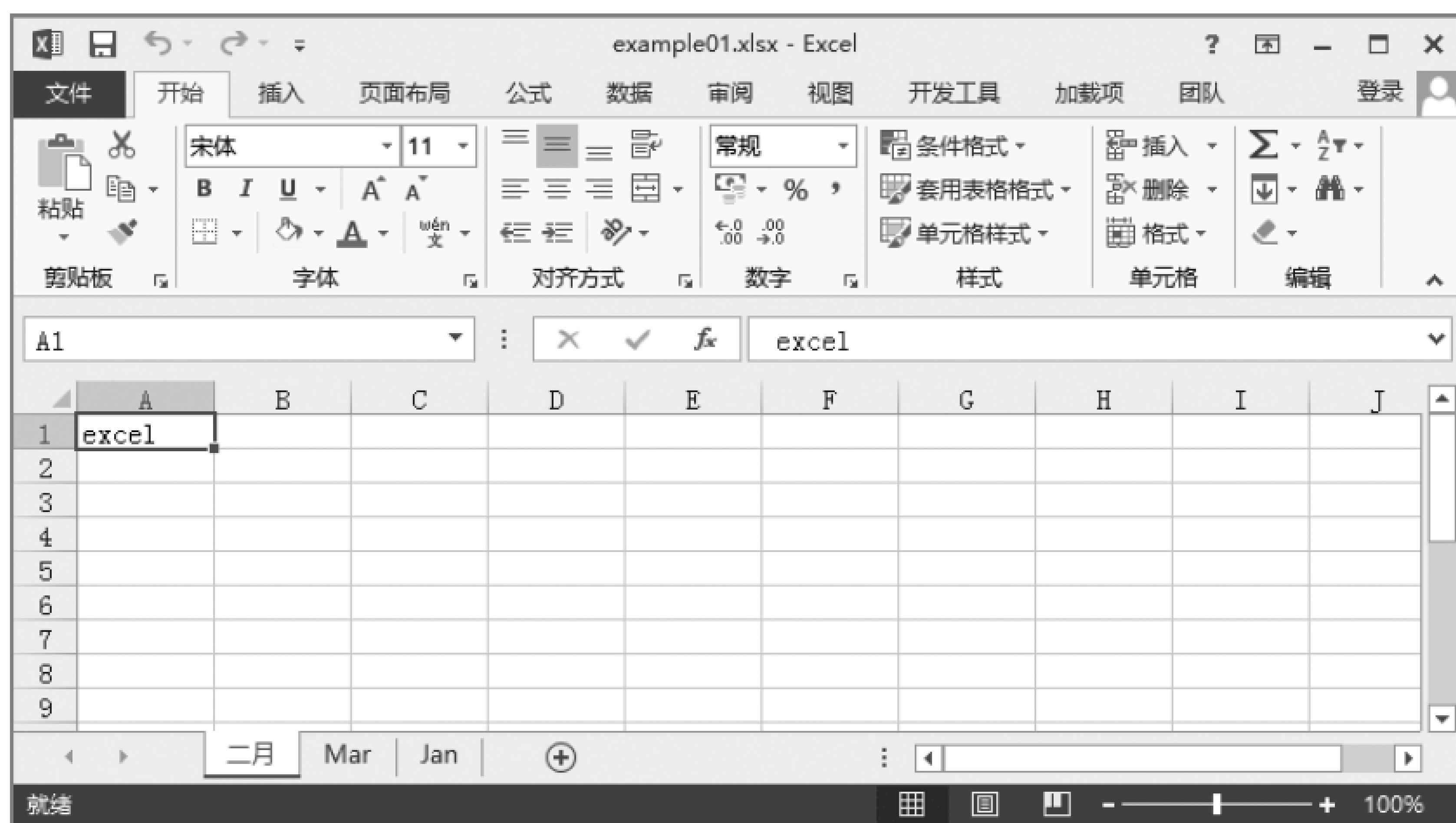


图 3-22　修改后 Excel 文件的内容

可以看到 Jan 工作表移到了最后，Feb 工作表被重命名。

Word、PowerPoint 文档也是压缩文件，可以用 WinRAR 查看和打开。以上演示的是手工使用 WinRAR 查看、修改 Office 文档，当然也可以用 Shell 调用 WinRAR 进行自动解压。

下面的过程把 example01.xlsx 文件中的 Workbook.xml 部分解压到 Destination 文件夹中。

```
    Sub 解压 Office 文档 ()
        Const WorkDir As String = "E:\"             '默认目录
        Dim WinrarExe As String                     'WinRAR 可执行文件路径
        Dim Command As String                       '命令、开关参数
        Dim RarFile As String                       '压缩包路径
        Dim Contents As String                      '文件、文件夹的路径
        WinrarExe = GetSetupPath("WinRAR.exe") & "\WinRAR.exe"
        Command = " E "
        RarFile = WorkDir & "example01.xlsx"
        Contents = WorkDir & "Destination"          '释放到 Destination 路径下
        Shell AddQuote(WinrarExe) & Command & AddQuote(RarFile) & " xl\workbook.xml "
& AddQuote(Contents), vbNormalFocus
    End Sub
```

如果是带有自定义功能区的文档，用压缩包打开后，里面有更多的内容，这些在稍后的章节讲解。

以上内容的源代码文件为“实例文档 10.xlsm”。

3.2 使用 Shell32 对象

Shell32 对象中的 Folder 对象与 FSO 文件系统对象中的 Folder 用法非常相似，都能代表计算机中的一个文件夹。但是，Shell32.Folder 还可以表示一个扩展名为 .zip 的压缩文件，而 FSO 不能操作到压缩文件的内部。

因此，本节介绍一下用 Shell32 对象处理计算机中的文件夹和文件以及 .zip 压缩包中的内容。

3.2.1 引入 Shell32 对象

前期绑定方式：在 VBA 工程中添加外部引用“ Microsoft Shell Controls And Automation”，如图 3-23 所示。

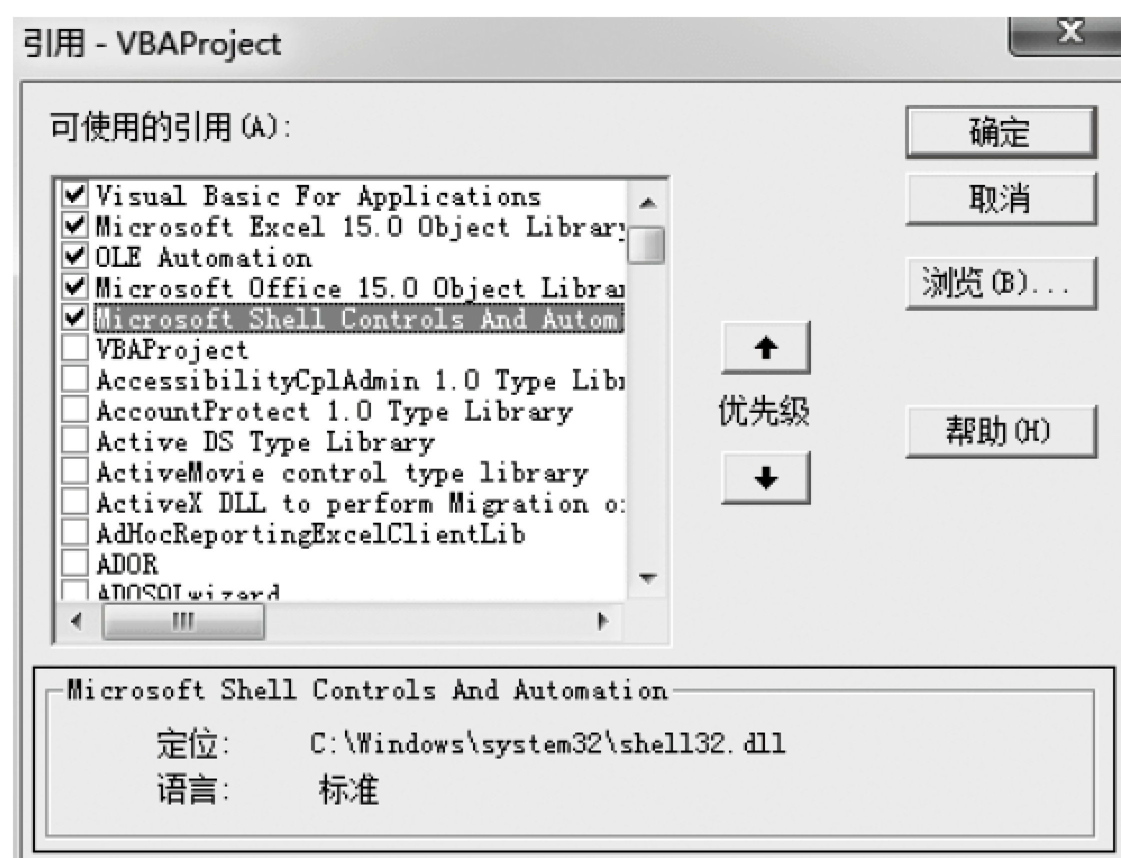


图 3-23 添加外部引用

代码中使用 Dim ShellApp As New Shell32.Shell 声明了一个新的 Shell32 的应用程序对象。后期创建对象的方法如下。

```
Set ShellApp = CreateObject("Shell.Application")
```

3.2.2　使用 namespace 返回文件夹

FSO 对象中，使用 GetFolder 返回一个 Folder 对象，而 Shell32 对象中，使用 namespace 返回一个 Folder 对象。

```
Sub Test1()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder
    With ShellApp
        Set fd = .Namespace("C:\temp")
        Debug.Print fd.Items.Item.Path  '返回文件夹的路径
        Debug.Print fd.Items.Count      '返回该文件夹中包含的内容个数(子文件夹+文件)
    End With
End Sub
```

以上过程中，fd 是一个文件夹对象变量，本例用它来指代 C:\temp 这个文件夹。

运行上述程序，在立即窗口打印文件夹的路径、子文件夹和文件的总数，运行结果如图 3-24 所示。

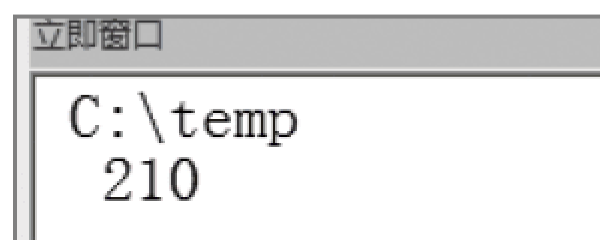


图 3-24　运行结果

上面的实例中，文件夹的规定是在 namespace 的参数中直接指定的，如果要让用户选择一个文件夹，则可以使用 BrowseForFolder 函数。

3.2.3　文件夹选择对话框

BrowseForFolder 函数的语法如下。

```
BrowseForFolder(Hwnd, Title, Options, RootFoler)
```

返回一个 Folder 对象。各参数说明如下。

- Hwnd：对话框的所属句柄，长整型。设置为 0 表示在桌面弹出对话框。
- Title：对话框显示的提示语。
- Options：对话框样式设定参数，十六进制数。
- RootFolder：文件夹选择对话框的起始路径，也可以是特殊的文件夹常量。

下面的代码在 Excel 上面弹出一个文件夹选择对话框，起始目录设置为宏所在工作簿的路径。

```
Sub Test2()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder
    With ShellApp
        Set fd = .BrowseForFolder(Hwnd:=Application.Hwnd, Title:="你必须选择一个
文件夹：", Options:=&H0, RootFolder:=ThisWorkbook.Path)
        If fd Is Nothing = False Then
```

```
            Debug.Print fd.Items.Item.Path
        End If
    End With
End Sub
```

代码分析：BrowseForFolder 函数弹出浏览文件夹对话框，用户选择文件夹并单击“确定”按钮，fd 会返回一个 Folder 对象，如果单击“取消”按钮，那么 fd 就是 Nothing。

运行上述程序，自动弹出一个选择文件夹的对话框，如图 3-25 所示。



图 3-25　浏览文件夹对话框

3.2.4　遍历文件夹中的内容

Shell32 中的 FolderItem 对象是指包含在文件夹中的文件或子文件夹。

假设 D:\Download 下的文件内容如图 3-26 所示。

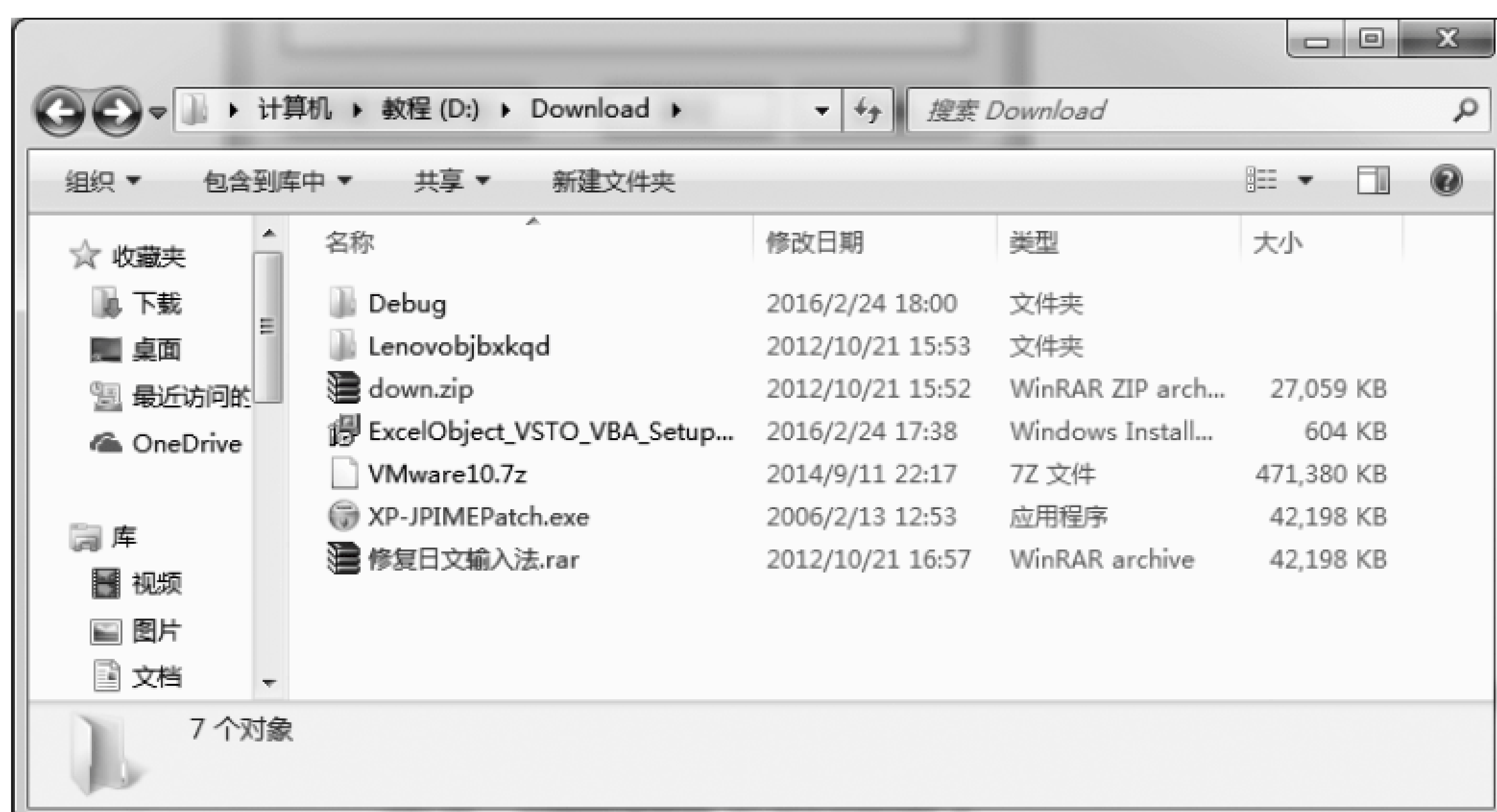


图 3-26　文件夹中的内容

文件夹中有 2 个子文件、5 个文件。

下面的代码遍历 Download 文件夹下所有项目的路径、类型。

```
Sub Test3()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, item As Shell32.
FolderItem
    With ShellApp
        Set fd = .Namespace("D:\Download")
        If fd Is Nothing = False Then
            Debug.Print fd.Items.Count '返回该文件夹中包含的内容个数(子文件夹+文件)
            For Each item In fd.Items
                Debug.Print item.Path, item.Type
            Next item
        End If
    End With
End Sub
```

运行上述程序，立即窗口打印出文件夹中的所有内容，如图 3-27 所示。

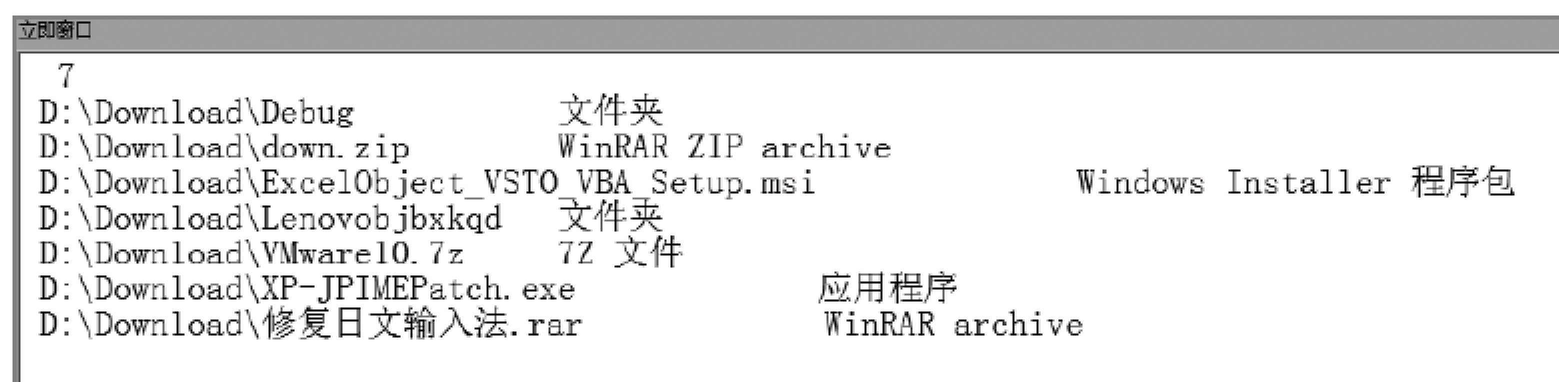

立即窗口
7
D:\Download\Debug　文件夹
D:\Download\down.zip　WinRAR ZIP archive
D:\Download\ExcelObject_VSTO_VBA_Setup.msi　Windows Installer 程序包
D:\Download\Lenovobjbxkqd　文件夹
D:\Download\VMware10.7z　7Z 文件
D:\Download\XP-JPIMEPatch.exe　应用程序
D:\Download\修复日文输入法.rar　WinRAR archive

图 3-27　遍历文件夹中的内容

可以看出，子文件夹的 Type 是“文件夹”。

如果要遍历所有子文件夹中的所有内容，需要用下面的递归函数。

```
Sub Recursion(ByVal Parent As Shell32.Folder)
    Debug.Print Parent.Title
    For Each f In Parent.Items
        If f.Type = " 文件夹 " Then
            Recursion f.GetFolder
        Else
            Debug.Print f.Path
        End If
    Next f
End Sub

Sub 遍历所有内容 ()
     Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, item As Shell32.
FolderItem
    With ShellApp
        Set fd = .Namespace("D:\Download")
        Recursion fd
    End With
End Sub
```

运行上面的“遍历所有内容”这个过程，立即窗口打印出 Download 文件夹下的所有内容（包含递归文件夹），如图 3-28 所示。

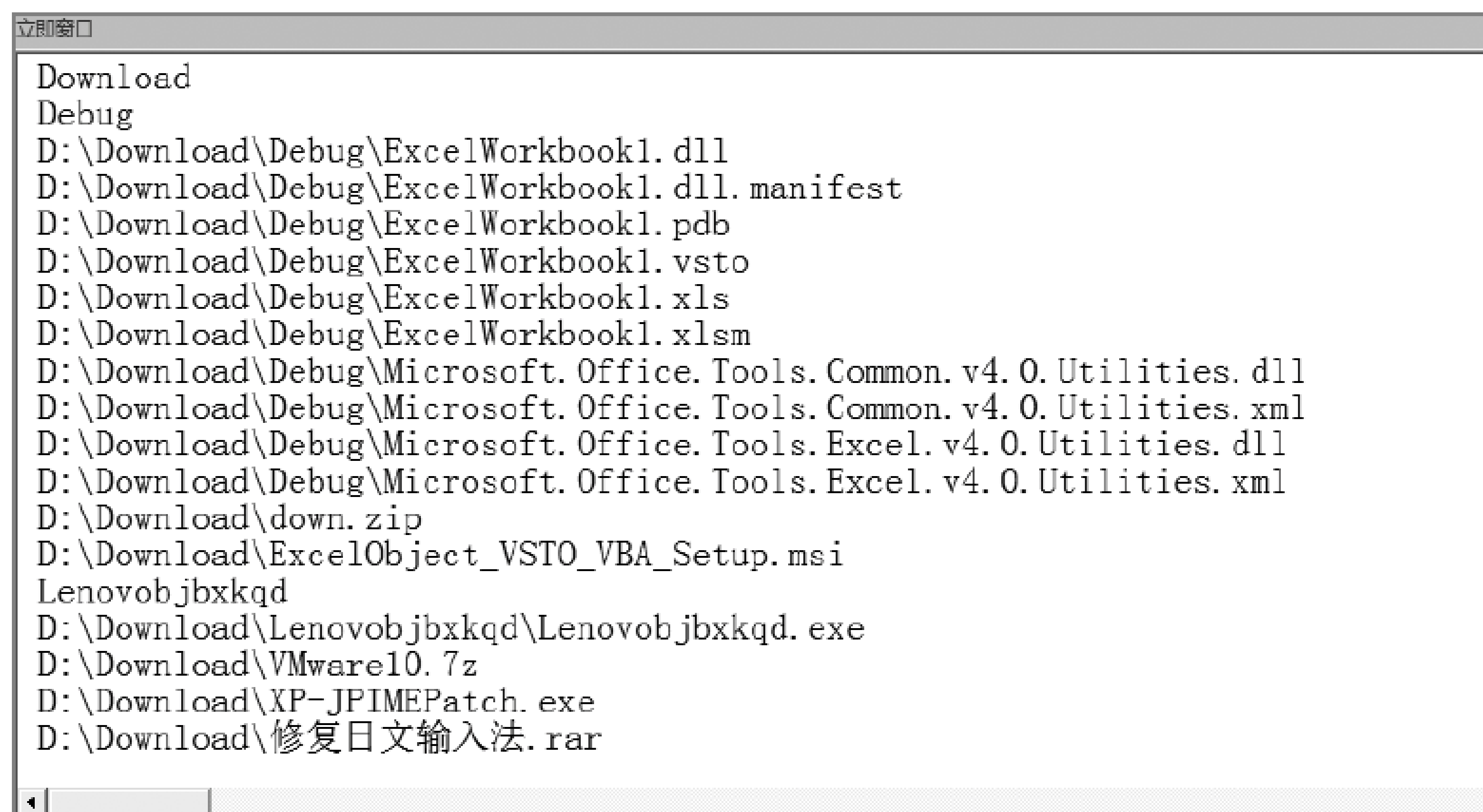

立即窗口
Download
Debug
D:\Download\Debug\ExcelWorkbook1.dll
D:\Download\Debug\ExcelWorkbook1.dll.manifest
D:\Download\Debug\ExcelWorkbook1.pdb
D:\Download\Debug\ExcelWorkbook1.vsto
D:\Download\Debug\ExcelWorkbook1.xls
D:\Download\Debug\ExcelWorkbook1.xlsm
D:\Download\Debug\Microsoft.Office.Tools.Common.v4.0.Utilities.dll
D:\Download\Debug\Microsoft.Office.Tools.Common.v4.0.Utilities.xml
D:\Download\Debug\Microsoft.Office.Tools.Excel.v4.0.Utilities.dll
D:\Download\Debug\Microsoft.Office.Tools.Excel.v4.0.Utilities.xml
D:\Download\down.zip
D:\Download\ExcelObject_VSTO_VBA_Setup.msi
Lenovobjbxkqd
D:\Download\Lenovobjbxkqd\Lenovobjbxkqd.exe
D:\Download\VMware10.7z
D:\Download\XP-JPIMEPatch.exe
D:\Download\修复日文输入法.rar

图 3-28　递归遍历文件夹中的所有内容

由于 namespace 不只限于文件夹，还能把 .zip 压缩包当成一个文件夹，因此下面讲述遍

历 .zip 压缩包中的内容。

3.2.5 遍历 .zip 压缩包中的内容

假设 D: 盘下有一个“东北三省 .zip”的压缩包，其内部文件结构如图 3-29 所示。



图 3-29 压缩包

运行下面的过程，可以把压缩包中的所有文件、路径列举出来。

```
Sub 遍历压缩包中内容 ()
     Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, item As Shell32.
FolderItem
    With ShellApp
        Set fd = .Namespace("D:\ 东北三省 .zip")
        Recursion fd
    End With
End Sub
```

其中，Recursion 是前面讲过的用于递归遍历的函数。

运行上述程序，立即窗口打印出压缩包中的所有内容，如图 3-30 所示。

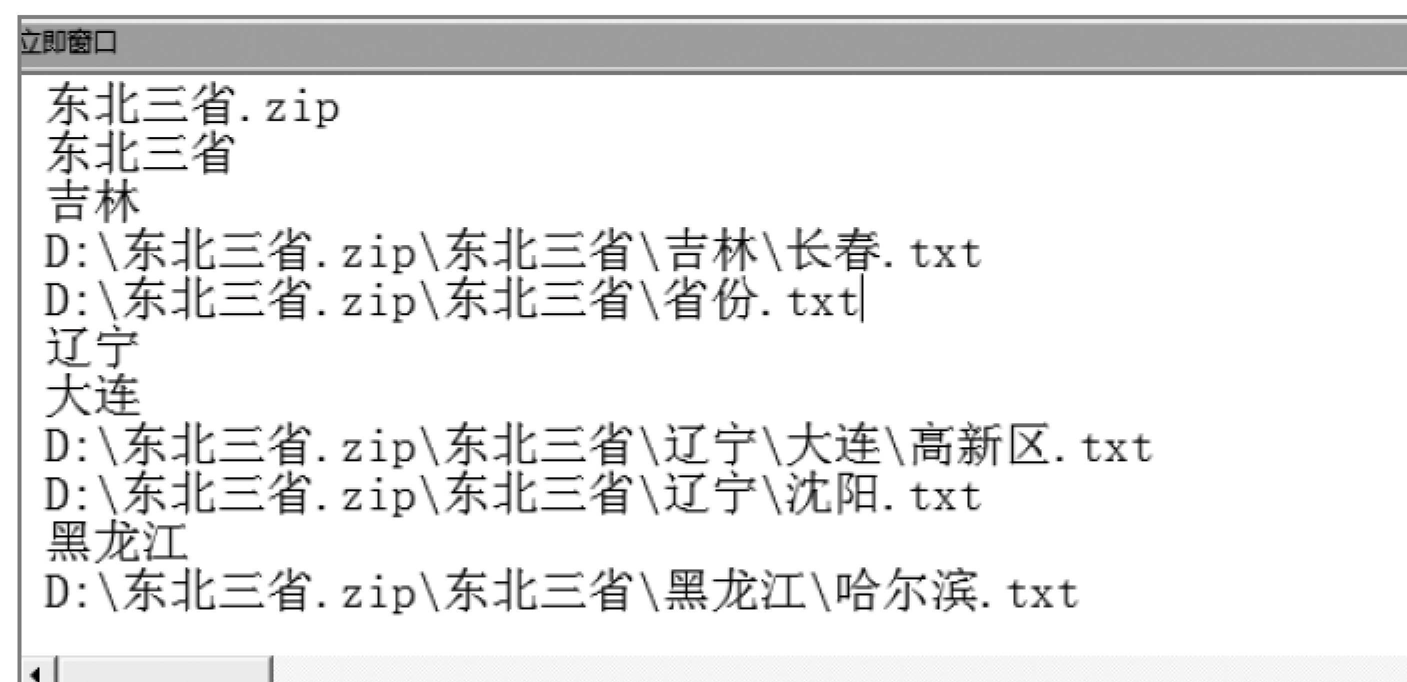


图 3-30 递归遍历压缩包中的所有内容

3.2.6 遍历 Office 文档中的内容

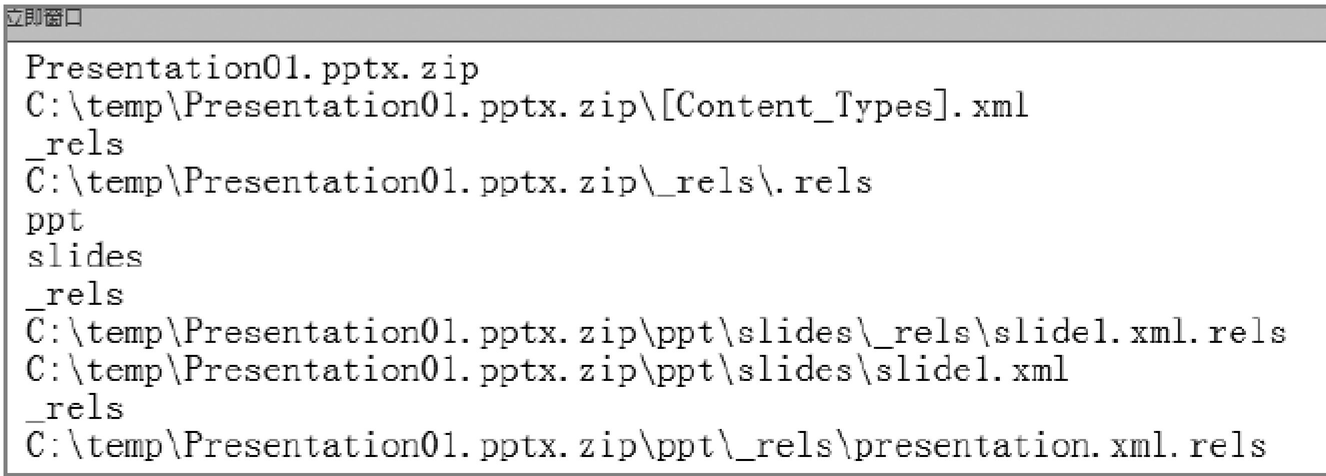
Office 文档的扩展名不是 .zip，理论上用 Shell32 无法遍历其内部内容，然而，可以先

把 Office 文档更改扩展名为 .zip，等遍历完后，再撤销重命名即可。

假设文件夹中有一个幻灯片文件 Presentation01.pptx。下面用 Shell32 遍历该文件的所有内部文件。

```
    Sub 遍历 Office 文档 ()
        Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, item As Shell32.
FolderItem
        Dim doc As String
        doc = "C:\temp\Presentation01.pptx"
        Name doc As doc & ".zip"
        With ShellApp
            Set fd = .Namespace(doc & ".zip")
            Recursion fd
        End With
        MsgBox "下面撤销重命名！ "
        Name doc & ".zip" As doc
    End Sub
```

该幻灯片的内部文件比较多，因此只打印出一部分结果，如图 3-31 所示。



```
立即窗口
Presentation01.pptx.zip
C:\temp\Presentation01.pptx.zip\[Content_Types].xml
_rels
C:\temp\Presentation01.pptx.zip\_rels\.rels
ppt
slides
_rels
C:\temp\Presentation01.pptx.zip\ppt\slides\_rels\slide1.xml.rels
C:\temp\Presentation01.pptx.zip\ppt\slides\slide1.xml
_rels
C:\temp\Presentation01.pptx.zip\ppt\_rels\presentation.xml.rels
```

图 3-31　递归遍历 PPT 文件中的所有内容

用 WinRAR 打开该幻灯片，如图 3-32 所示。

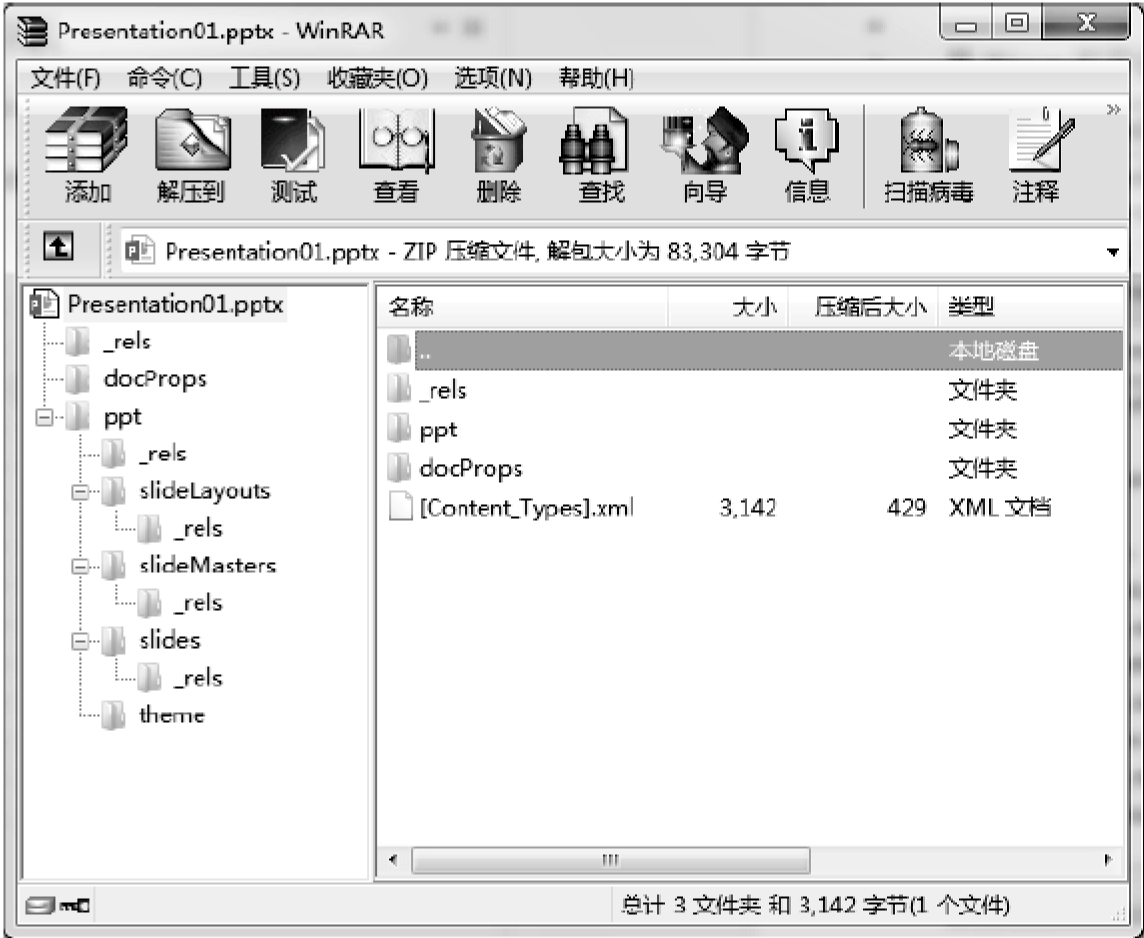


图 3-32　使用压缩包查看 PPT 文件的构造

3.2.7　CopyHere 方法

Shell32 中的 Folder 对象有 CopyHere 方法和 MoveHere 方法，作用是把其他地方的文件

（夹）复制或移动到 Folder 中。

下面的程序把 dist 路径下的所有文件、子文件夹复制到 temp 路径下。

```
Sub 复制文件到文件夹中 ()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, data As Shell32.Folder
    With ShellApp
        Set fd = .Namespace("C:\temp\")
        Set data = .Namespace("C:\dist\")
        fd.CopyHere data.Items
    End With
End Sub
```

代码分析：data.Items 表示该命名空间（路径）下的所有项目，包括文件、子文件夹。

如果要复制个别的项目，需要用 Item 属性来约定。例如把上面最后一行代码修改为如下形式。

```
fd.CopyHere data.Items.item("aaa\ 使用说明 .txt")
```

上述代码的含义是把 C:\dist\aaa\ 使用说明 .txt 这个文件直接复制到 temp 文件夹下。其中，aaa 是 dist 路径下的一个文件夹。

3.2.8 MoveHere 方法

MoveHere 方法与 Copy 方法几乎是一样的语法，唯一不同的是，使用 MoveHere 方法是把文件或文件夹移动到 Folder 中，也就相当于文件的移动、剪切。

下面举一个把文件夹中的文件移动到压缩包中的实例。首先在桌面或者任意文件夹中新建一个“WinRAR ZIP archive”空白压缩包，并把该压缩包重命名为 blank.zip，如图 3-33 所示。



图 3-33 手工新建一个空白压缩包

路径 C:\temp\datas 下有一百多个文本文件，如图 3-34 所示。

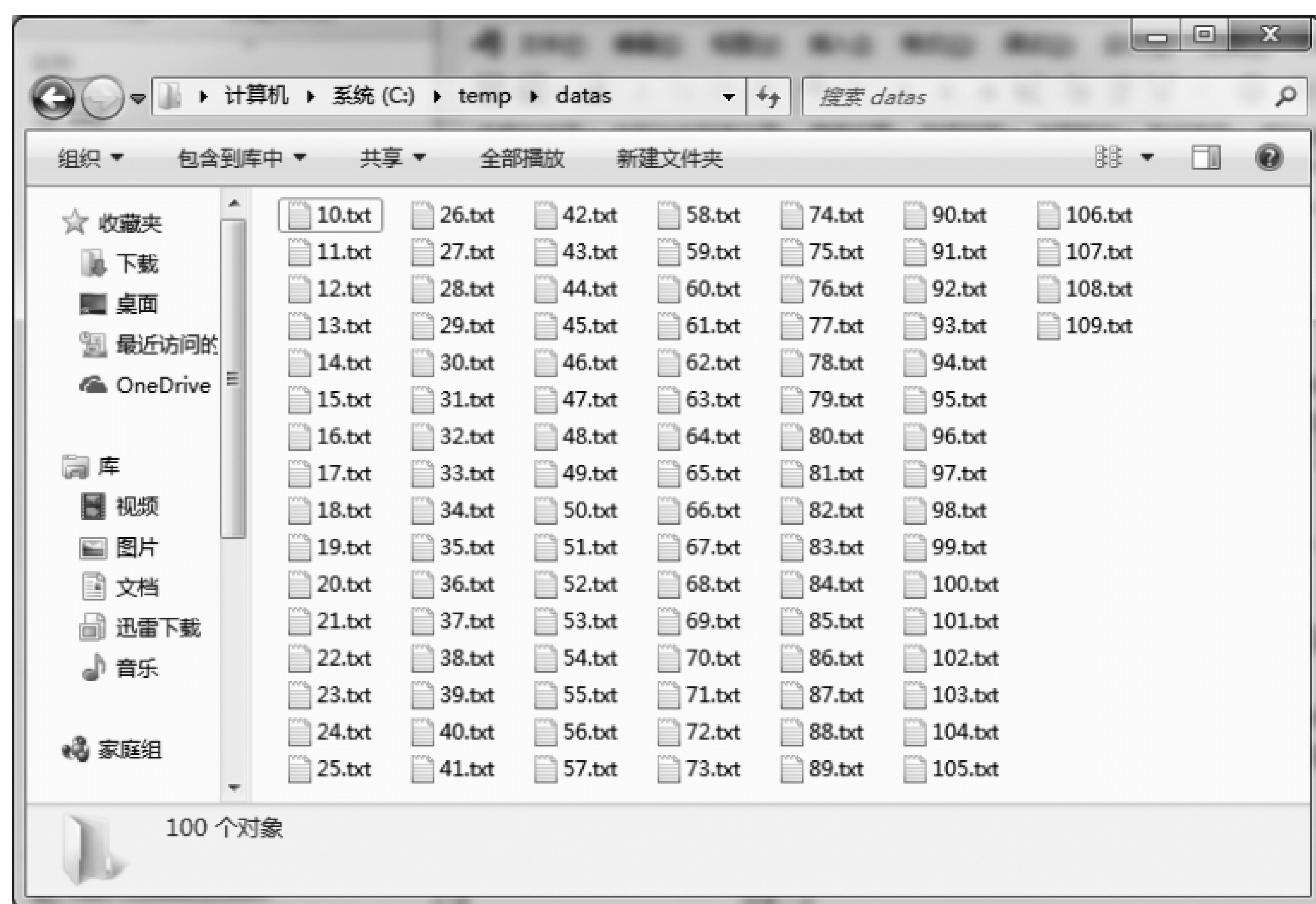


图 3-34　文件夹中包含多个文本文件

下面的程序把 datas 文件夹连同其所有子文件压缩到 blank.zip 中。

```
Sub 移动文件到压缩包中()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, data As Shell32.Folder
    With ShellApp
        Set fd = .Namespace("C:\dist\Blank.zip")
        Set data = .Namespace("C:\temp\datas")
        fd.MoveHere data
    End With
End Sub
```

执行程序后，使 WinRAR 软件查看压缩包 blank.zip 中的内容，如图 3-35 所示。

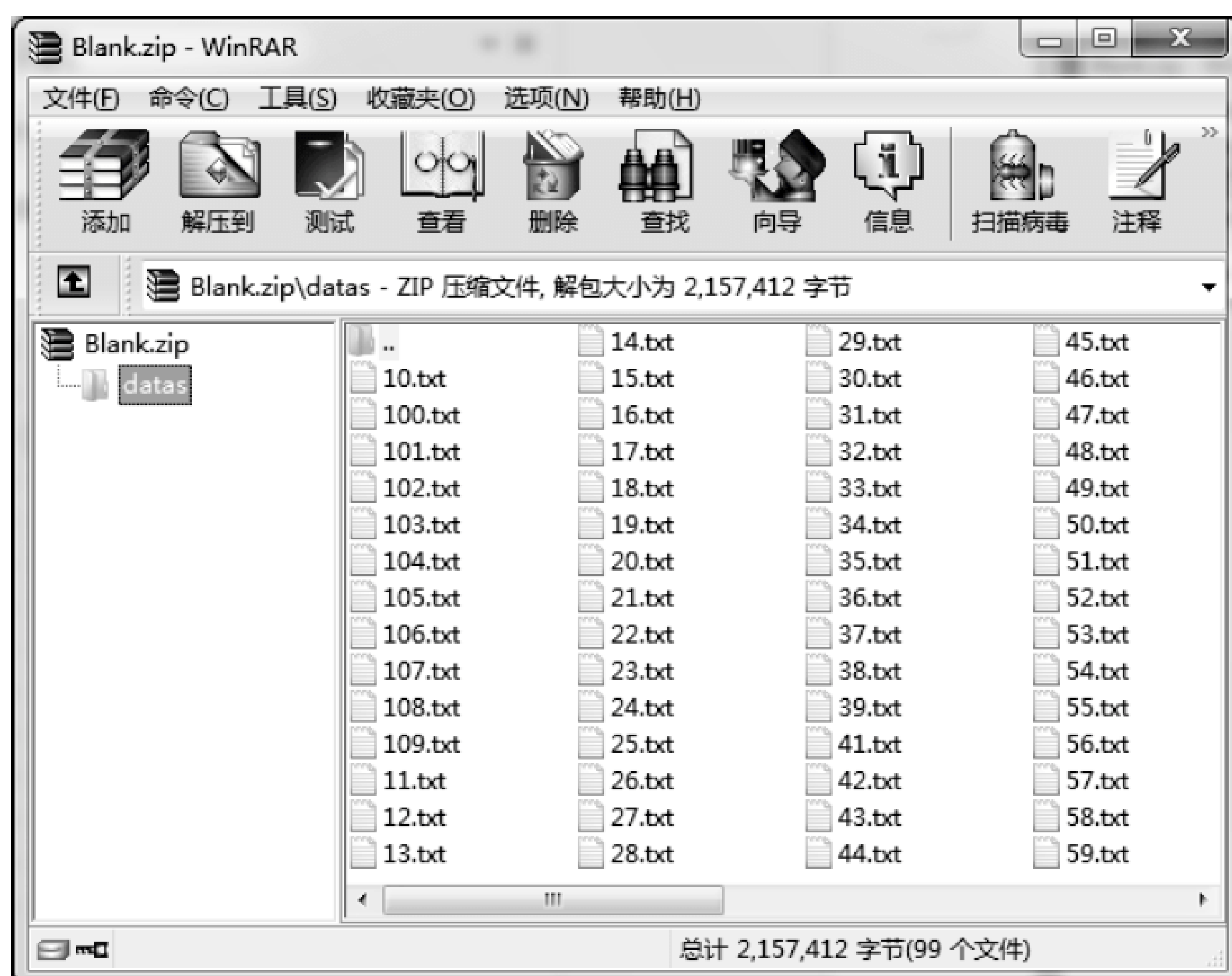


图 3-35　自动压缩文件夹

需要注意的是，如果最后一行代码修改为如下形式。

```
fd.MoveHere data.Items
```

执行的效果是，datas 下面所有的文本文件都直接压缩进去，而没有 datas 这个文件夹！

如果修改为 fd.MoveHere data.Items.Item("39.txt")，则只把一个文本文件移动到压缩包的根目录下。

反过来，CopyHere 方法、MoveHere 方法也可以从压缩包中释放内容到文件夹中。

下面的程序把上述有内容的 Blank.zip 中的 datas\40.txt 文件移动到 C:\lib 路径下。

```
Sub 释放压缩包中内容到文件夹 ()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, LIB As Shell32.Folder
    With ShellApp
        Set fd = .Namespace("C:\dist\Blank.zip\datas")
        Set LIB = .Namespace("C:\lib")
        LIB.MoveHere fd.Items.item("40.txt")
    End With
End Sub
```

代码分析：namespace 允许在压缩包路径后追加子路径，因此对象变量 fd 表示的是压缩包中的 datas 文件夹。

LIB.MoveHere fd.Items.item("40.txt") 把 fd 中的项目移动到 LIB 中，一定不要搞错方向。

另外，除了上述讲过的用鼠标右键新建 .zip 压缩包之外，也可以用代码自动在路径下产生一个空白的 .zip 压缩包。

```
Sub 新建空白 zip 压缩包 ()
    Open "C:\Container.zip" For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
End Sub
```

运行上述过程，会在 C: 盘下产生 Container.zip，这个压缩包里没有任何内容。

3.2.9 处理文件覆盖

使用 CopyHere 方法、MoveHere 方法进行文件复制、移动时，如果目的地已经存在名称相同的文件，执行程序过程中会弹出是否复制、替换的对话框，如图 3-36 所示。



图 3-36　存在同名文件的询问对话框

如果要默认强制替换已存在文件，屏蔽该对话框，可以在方法之后加一个 vOptions 参数，并设置为 16。例如：

```
fd.CopyHere vItem:=data.Items.item ("aaa\使用说明.txt"), vOptions:=16
```

这样，运行到这行代码时，即使存在同名文件，也强制替换。

3.2.10　处理异步问题

使用 CopyHere、MoveHere 方法移动内容时，不管移动操作是否已完成，VBA 代码都会继续向下执行。如果移动的文件容量越大，异步问题越明显。对于一些要求苛刻的程序任务，有必要让程序同步压缩进度。

为了能够让 VBA 识别到文件移动的进度，需要在 CopyHere、MoveHere 方法之后加一个 Do…Loop 循环，循环跳出的条件是目标压缩包中的文件总数达到一个指标。

假设 E:\Joker 路径下有 52 张扑克牌图片，使用下面的程序把这 52 个 .jpg 格式的图片全部移动到新建的压缩包 C:\temp\Package.zip 中。

```
Sub 处理异步问题 ()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, data As Shell32.Folder
    Open "C:\temp\Package.zip" For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
    With ShellApp
        Set fd = .Namespace("C:\temp\Package.zip")
        Set data = .Namespace("E:\Joker")
        fd.MoveHere data.Items, 16
    End With
    Do Until fd.Items.Count = 52
        Application.Wait Now() + TimeValue("00:00:01")
    Loop
    MsgBox " 移动操作已完成 !", vbInformation
End Sub
```

代码分析：首先创建一个空白 .zip 压缩包，该压缩包中项目个数为 0。其次使用对象变量 fd 来指代该压缩包，然后把 Joker 文件夹下的所有文件移动到压缩包中，移动过程中 fd.Items.Count 一定小于 52，因此根据这个特征可以让 VBA 代码阻塞在 Do 循环内。最后，压缩操作结束，跳出 Do 循环，弹出对话框，如图 3-37 所示。



图 3-37　压缩操作完成才弹出对话框

3.2.11　修改 Office 文档功能区

Office 2007 以上版本创建的文档允许自定义功能区。自定义功能区的 XML 代码存储在文档中 CustomUI 文件夹下，文件名一般为 customUI14.xml。本书源代码中的 example02.xlsx 文件用 WinRAR 打开后，可以看到自定义功能区的部分，如图 3-38 所示。

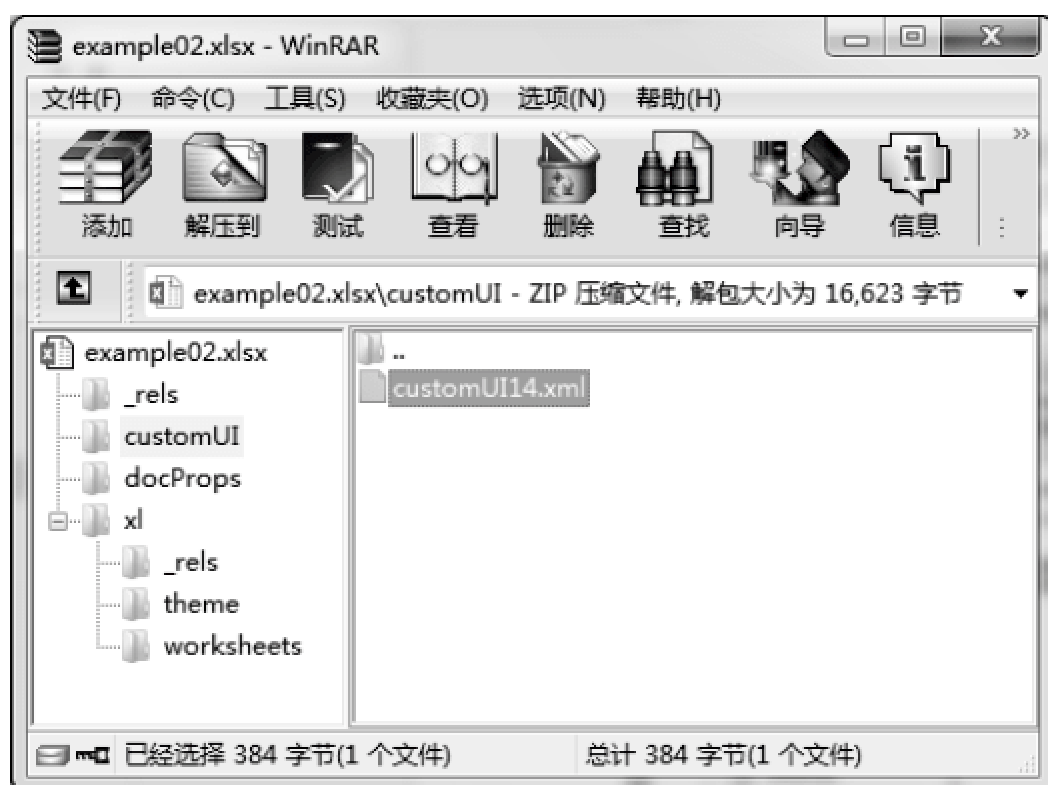


图 3-38 Excel 文件的内部构造

双击 customUI14.xml，可以看到 XML 代码，如图 3-39 所示。

```
- <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  - <ribbon startFromScratch="false">
    - <tabs>
      - <tab id="exampleID1" label="RibbonXmlEditor">
        - <group id="GroupID2" label="Author:ryueifu">
            <button id="ButtonID3" label="原始按钮" imageMso="ChartTypeOtherInsertGallery" size="large" />
          </group>
        </tab>
      </tabs>
    </ribbon>
  </customUI>
```

图 3-39 customUI 代码

如果在 Excel 中打开该文件，如图 3-40 所示。

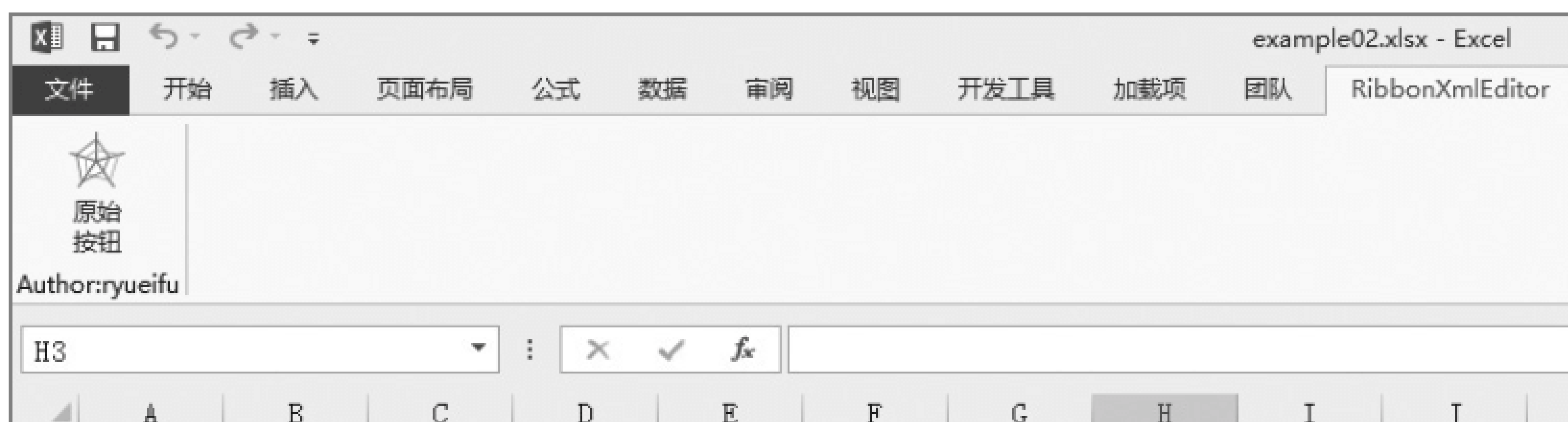


图 3-40 包含 customUI 部分的 Excel 文件

下面通过 Shell32 的 MoveHere 方法更改 XMl 代码，从而更改功能区的外观显示。

具体实现原理和步骤如下。

（1）在工作簿关闭的前提下，后面追加 .zip 扩展名，以便 Shell32 访问。

（2）使用 MoveHere 方法把 customUI14.xml 文件移动到某个文件夹中。

（3）使用 XML 外部对象自动修改文件夹中的 customUI14.xml 文件。

（4）用 MoveHere 方法把文件夹中的 customUI14.xml 文件逆向移动回到压缩包中的 customUI 文件夹下。

（5）删掉工作簿后面的 .zip，恢复为正常的 Excel 工作簿。具体代码如下。

```
Sub 修改 Office 文档功能区 ()
    Dim ShellApp As New Shell32.Shell, fd As Shell32.Folder, temp As Shell32.Folder
    Dim wbk As String
```

```
    Dim X As New DOMDocument              '引用Microsoft XML v6.0
    Dim Ribbon As String
    wbk = "E:\处理压缩文件\example02.xlsx"
    Name wbk As wbk & ".zip"              '暂时重命名为.zip压缩包
    With ShellApp
        Set fd = .Namespace(wbk & ".zip\customUI") '定位到压缩包中CustomUI文件夹
        Set temp = .Namespace("C:\temp\")
        temp.MoveHere fd.Items.item("customUI14.xml"), 16
                                          '移动功能区代码到磁盘文件夹中
        If X.Load("C:\temp\customUI14.xml") Then       '装载xml文件并适当替换
            Ribbon = Replace(X.XML, "原始按钮", "被我修改")
            Ribbon = Replace(Ribbon, "ChartTypeOtherInsertGallery", "M")
            If X.LoadXML(Ribbon) Then
                X.Save "C:\temp\customUI14.xml"        '保存被修改的xml文件
            End If
        End If
        fd.MoveHere temp.Items.item("customUI14.xml"), 16
                                          '把被修改的xml文件压缩回工作簿中
        Stop
    End With
    Name wbk & ".zip" As wbk              '恢复原来的扩展名
End Sub
```

代码分析：为了确保压缩操作完成后再重命名，在重命名代码之前加一个 Stop 语句。

运行上述代码后，在 Excel 2013 中打开 example02.xlsx 文件，会看到功能区中的按钮标题和图标都发生了变化，如图 3-41 所示。

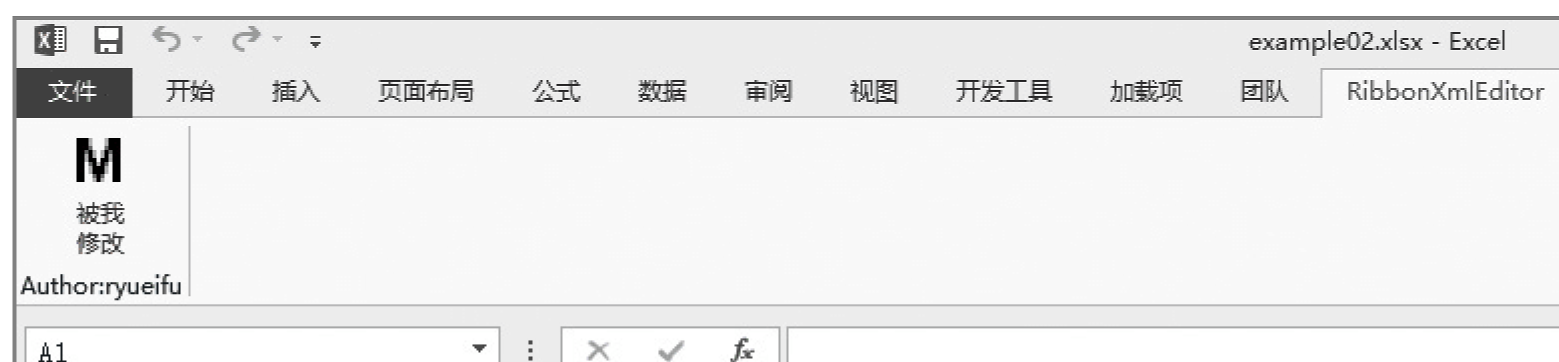


图 3-41　使用 Shell32 修改 Excel 文件的 customUI部分

以上这部分知识是自定义功能区的铺垫，同时表明可以从压缩文件的角度去研究和处理 Office 文档。

以上内容的源代码文件为“实例文档 11.xlsm”。

3.3　本章小结

Shell 调用 WinRAR 处理压缩包，Shell32 对象处理压缩包，这些操作都是异步的，也就是说，VBA 代码在不知道压缩操作是否已完成的情况下继续执行后面的代码。因此，当使用 CopyHere、MoveHere 方法之后，需要补充一些监测压缩操作进度的代码，防止后续的 VBA 代码过早执行。

第 4 章

操作 XML

XML（Extensible Markup Language）是指可扩展性标记语言。1998 年 2 月，W3C 正式批准了可扩展性标记语言的标准定义。可扩展性标记语言可以对文档和数据进行结构化处理，从而能够在部门、客户和供应商之间交换，实现动态内容生成、企业集成和应用开发。

通俗地讲，XML 是一种用标签描述的数据格式、文件格式。XML 本身不是编程语言，是一种数据传输格式。XML 文件可以存储各种信息，例如员工信息、商品信息等。

XML 具有内容和格式相互分离、可扩展性等优势，即使是微软也大量采用 XML 技术来开发 Office。前面讲过，Office 文档经解压后可以看出也是由大量 XML 文件组成的。此外，微软 Office 采用 XML 语言来描述 Office 界面，也允许用户对 Office 界面进行自定义。因此，了解和掌握 XML 语法对于 Office VBA 开发具有非常重要的意义。

本章首先讲述 XML 的基础知识和语法规则，然后重点讲述 DOMDocument 对象模型，以及使用 DOMDocument 对象读写和修改 XML 的技术。

本章用到的外部引用和重要对象：

- Microsoft XML, v6.0
  - MSXML2.DOMDocument60

4.1 XML 构成

一个典型的 XML 文件一般由处理指令（XML 声明）、节点树（NodeTree）和一些必要的注释（Comments）构成。

图 4-1 所示是一个典型的 XML 文件（文件名为“西南省份 .xml”）用 NotePad++ 打开后的效果。

第 1 行是该 XML 文件的处理指令。

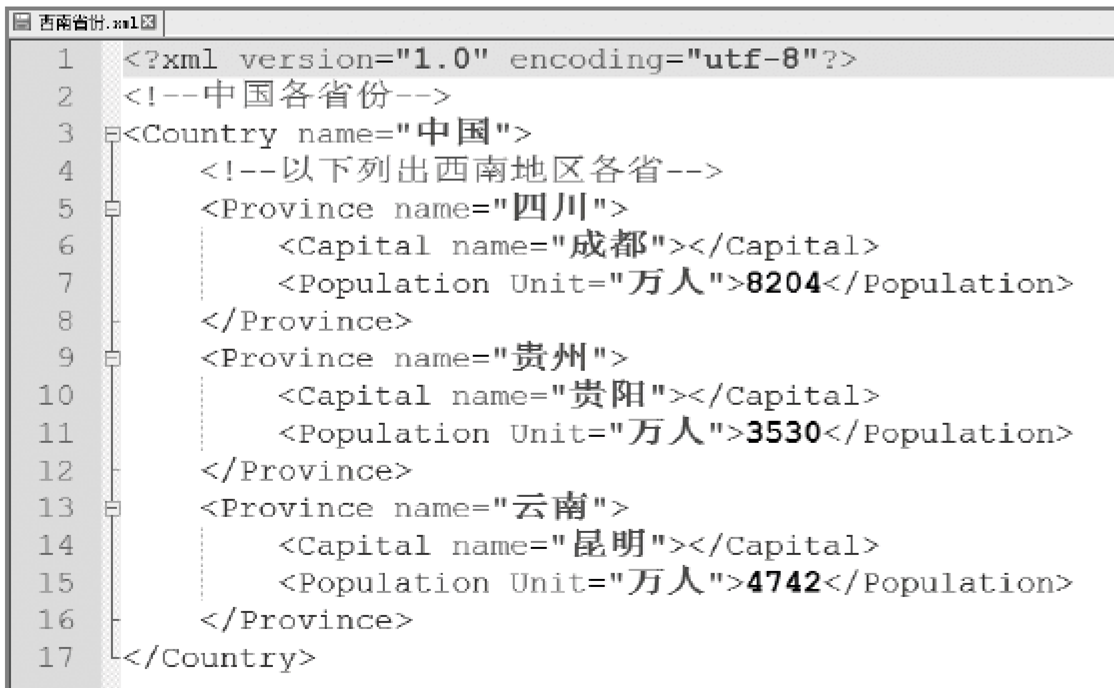
在第 2 行和第 4 行中，<!-- 与 --> 包起来的部分是注释。

其余各行用 <> 包起来的部分就是 XML 中的标签，也就是元素节点。

该文件描述的是中国西南各省的首府城市、各省人口总数这些信息。

XML 文档模型中，处理指令、注释、元素、文本内容等都属于 XML 的节点，换句话说，图中的 17 行，每行都是节点（Node）。

上述 XML 文件如果用 Treeview 控件绘制一个示意图，将会是如图 4-2 所示的样子。



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!--中国各省份-->
3  <Country name="中国">
4      <!--以下列出西南地区各省-->
5      <Province name="四川">
6          <Capital name="成都"></Capital>
7          <Population Unit="万人">8204</Population>
8      </Province>
9      <Province name="贵州">
10         <Capital name="贵阳"></Capital>
11         <Population Unit="万人">3530</Population>
12     </Province>
13     <Province name="云南">
14         <Capital name="昆明"></Capital>
15         <Population Unit="万人">4742</Population>
16     </Province>
17 </Country>
```

图 4-1　“西南省份 .xml”源文件

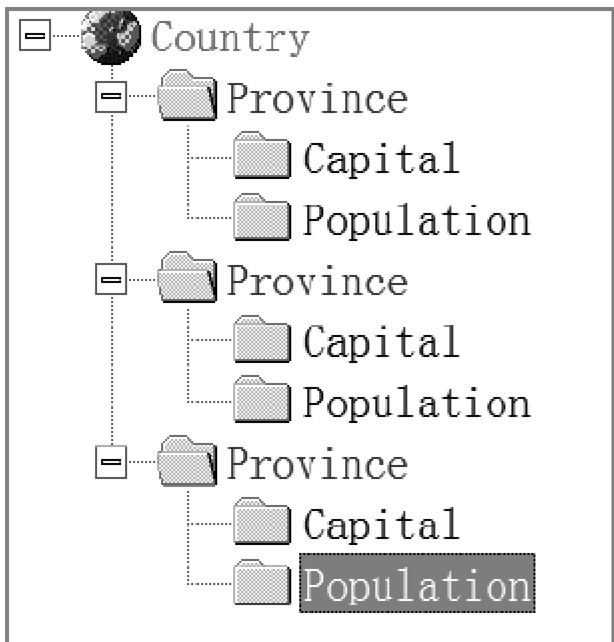


图 4-2　XML 文件结构示意图

可以看出，该 XML 的根元素节点是 Country，它下面管辖 3 个 Province 元素节点，每个 Province 下面有 Capital 和 Population 元素节点。

XML 可以包含很多种类型的节点，下面分别介绍最常见的节点类型。

4.1.1　元素节点

元素（Element）节点，是 XML 文件的骨架，处在数据存储的核心位置。与 HTML 语言中的标签类似，元素节点必须用标签符号括起来。

元素节点必须指定元素的名称，例如 <Staff></Staff> 就构成了一个完整的元素节点，这个元素的名称是 Staff。一般情况下，一个元素节点由开始标签和结束标签形成一个闭合环境。例如：

```
<Staff>
……Something……
</Staff>
```

也就是说，如果在 XML 文件的某个位置出现了 <Staff>，在其后面一定有个 </Staff> 与其呼应，这个结束标签的作用表示该标签就这么大一个作用范围。

元素节点可以包含文本内容、注释、子节点等，也可以什么都不包含。

例如，下面的 XML 代码段中，Staff 元素节点下面包含一个子元素节点：张三、一个文本内容节点（内容是：技术部），还包含一个注释节点（内容是：部分员工）。

```
<Staff>
    <张三>
    </张三>
技术部
<!-- 部分员工 -->
</Staff>
```

这里可以得出一个结论：元素节点可以是其他节点的父节点，同时自身也可以是其他节点的子节点。

需要注意的是，如果某元素节点下面没有包含任何子节点，其结束标签可以合并到开始标签中。例如前面的张三元素节点没包含任何子节点，两行可以合并为 <张三 />，如下所示。

```
<Staff>
    <张三 />
技术部
<!-- 部分员工 -->
</Staff>
```

这种元素节点的合并方式通常用于末梢元素中。

4.1.2 元素的属性

一个元素节点，可以规定若干属性（attributes），每个属性都由属性名称和属性值组成。属性与属性之间由一个以上的空格隔开。无论是任何类型的属性值，都必须用引号括起来。

元素属性的语法格式如下。

```
<元素名称 属性 1="value1" 属性 2="value2">
```

一个元素的所有属性用空格隔开，并且这些属性一定和元素的开始标签放在同一个 <> 里面。例如下面这句。

```
<Population Unit="万人">8204</Population>
```

这个元素节点 Population 有 1 个属性，属性名是 Unit，属性值是“万人”。比较容易混淆的是元素的属性和元素的内容节点，元素的属性一定放在元素的开始标签中，而文本内容通常是夹在元素的开始标签与结束标签之间，例如，本例中的数字 8204 就是元素 Population 的一个子节点，元素的文本内容不需要加引号。

元素的属性值既可以用双引号括起来，也可以用单引号括起来。

4.1.3 节点关系

要学好 XML，首先，学会判断节点的类型和作用；其次，关注节点之间的关系。XML 的节点树看起来错综复杂，实际上，节点和节点之间就两种关系：所属关系（父子关系，Parent-Child）与并列关系（兄弟关系，Sibling）。正是因为所属关系和并列关系的存在，才形成了节点树。

如果以“西南省份 .xml”为例，Country 根元素节点与各个 Province 都是父子关系，而 Province 之间是兄弟关系。同一个 Province 里面的 Capital 和 Population 之间也是兄弟关系。

4.1.4 文本节点

文本节点（TextNode）一般是指位于元素节点的起始标签与结束标签之间的文本内容。

下面的 XML 描述了两种早餐的名称、价格、详细信息和能量。

```
<breakfast>
    <food>
        <name>French Toast</name>
        <price>$4.50</price>
        <description>thick slices made from our homemade sourdough bread</description>
        <calories>600</calories>
    </food>
    <food>
        <name>Homestyle Breakfast</name>
        <price>$6.95</price>
        <description>two eggs, bacon or sausage, toast, and our ever-popular
hash browns</description>
        <calories>950</calories>
    </food>
</breakfast>
```

例如，<name>French Toast</name> 中的“ French Toast”就是一个文本节点，该节点是 <name> 元素节点的子节点。同理，“$4.50”这个文本节点的父节点是 <price> 元素节点。

4.1.5　注释节点

XML 中的注释（Comment）内容要放在 <!-- 与 --> 之间，一条注释也构成了一个节点。

```
<food>
    <name>French Toast</name>
    <!--This is my favorite food-->
    <price>$4.50</price>
    <description>thick slices made from our homemade sourdough bread</description>
    <calories>600</calories>
</food>
```

例如，上述 XML 中的 <!--This is my favorite food--> 是一个注释节点，其父节点是 <food> 元素节点。兄弟节点有 <name>、<price> 等元素节点。

4.1.6　处理指令节点

处理指令（Processing Instruction）一般书写于 XML 最顶端、根元素节点上方的部分。

如图 4-1 所示，顶部的 <?xml version="1.0" encoding="utf-8"?> 就是处理指令节点，里面包含了 XML 的版本、编码方式等信息。

处理指令节点是文档（DOMDocument）的子节点，与根元素（DocumentElement）节点是兄弟关系。

4.2 XML 语法规则

XML 文件与记事本文件不一样，如果不按规则书写 XML，那么得到的文件就是不合法的或者形式不良好的文件。

4.2.1 标签必须正确关闭

这里提到的标签，一般指元素节点的开始标签、结束标签。例如下面的语句来描述个人信息。

```
<person name="kitty" age="25">
```

这个元素节点只有开始标签，没有正确关闭，有如下两种修改方法。

```
<person name="kitty" age="25"/>
```

或者

```
<person name="kitty" age="25"></person>
```

4.2.2 严格区分大小写

开始标签与结束标签必须是相同的内容。

```
<staff>
    <person name="kitty" age="25"/>
    <person name="allen" age="27"/>
</STAFF>
```

上面的根元素 <staff> 与结束标签中的单词不对应，因此不是一个合法的 XML。

4.2.3 必须有根元素

一个 XML 文件有且只有一个根元素（DocumentElement），该元素节点是整个文档的最顶层，这个根元素可以有很多子节点，但是不能有兄弟元素。

例如图 4-1 中的 <Country> 就是整个文档的根元素。

4.2.4 父子元素必须正确嵌套

作为一个父节点，其开始标记应位于所有子节点之前，结束标记应位于所有子节点之后。

```
<Parent>
    <Child name="kitty" age="25">
    </Child>
</Parent>
```

以上是一个正确的父子节点嵌套，下面是错误的嵌套方式。

```
<Parent>
    <Child name="kitty" age="25">
    </Parent>
</Child>
```

4.2.5　属性值必须加引号

属性值与文本内容节点不同，属性值必须加引号。

```
<Parent>
    <Child name="kitty" age=25>
    </Child>
</Parent>
```

以上 XML 中的 age 属性值未加引号，因此不合法。应改为：age="25"。

4.3　查看和编辑 XML

XML 文件与 HTML 网页文件类似，可以用记事本程序创建、编辑，可以用网页浏览器查看 XML 文件的效果。

更专业的、具有针对性的 XML 编辑器还有 FrontPage、XMLNotepad、XMLSpy 等软件。

4.3.1　使用记事本程序创建 XML 文件

首先打开记事本程序，新建一个空白文档，输入 XML 代码。然后关闭并保存这个文本文件。最后把这个文本文件的扩展名修改为 .xml 即可，如图 4-3 所示。

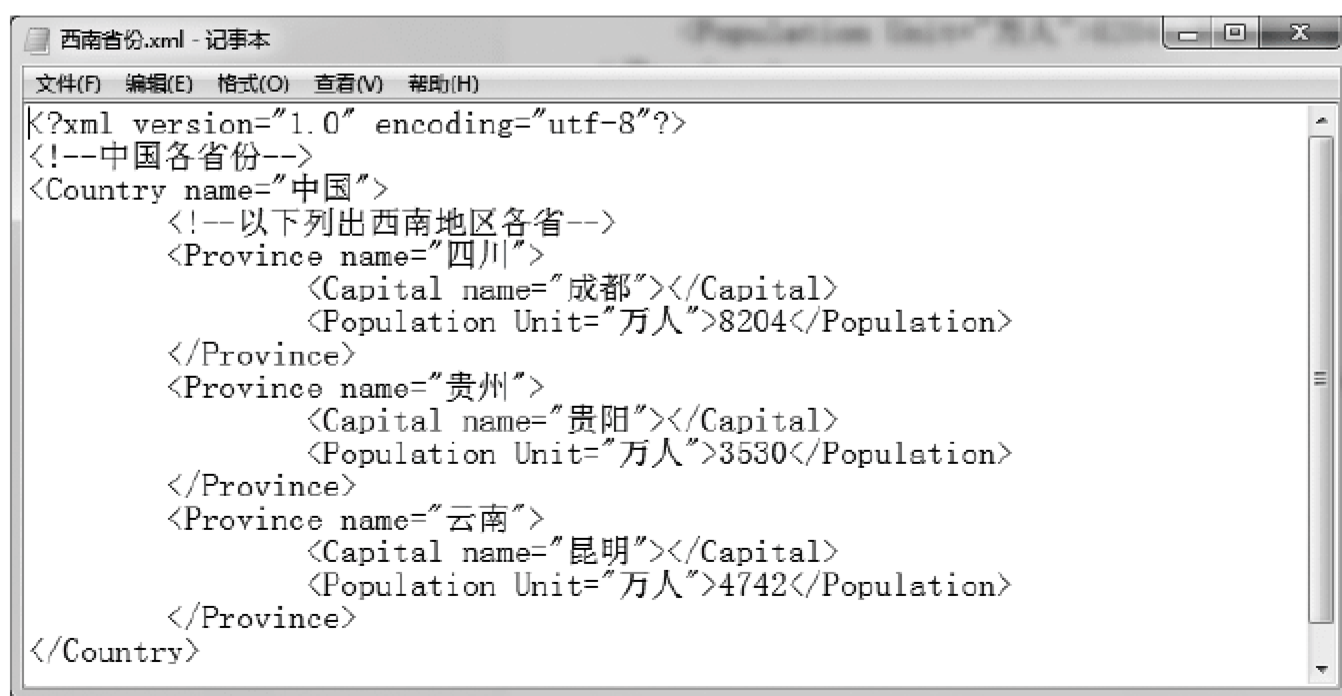


图 4-3　用记事本程序编辑 XML 文件

注意　编辑过程中，尽量保持良好的缩进，避免不必要的空白行。尽管不恰当的缩进、多余的空白行不会影响 XML 的结构。

4.3.2　使用 WebBrowser 控件显示 XML

VBA 编程中，可以借助 WebBrowser 控件发挥浏览器的作用，该控件可以在 VBA 窗体上显示网页、XML 文件以及 .gif 图片等。

在 VBA 的用户窗体的控件工具箱上右击，在弹出菜单中选择“附加控件”，弹出附加控件对话框。

在附加控件对话框中勾选“Microsoft Web Browser”或者“Microsoft Internet Control”，工具箱中会出现一个“地球”图标的控件。拖动该控件到 UserForm 上即可，如图 4-4 所示。

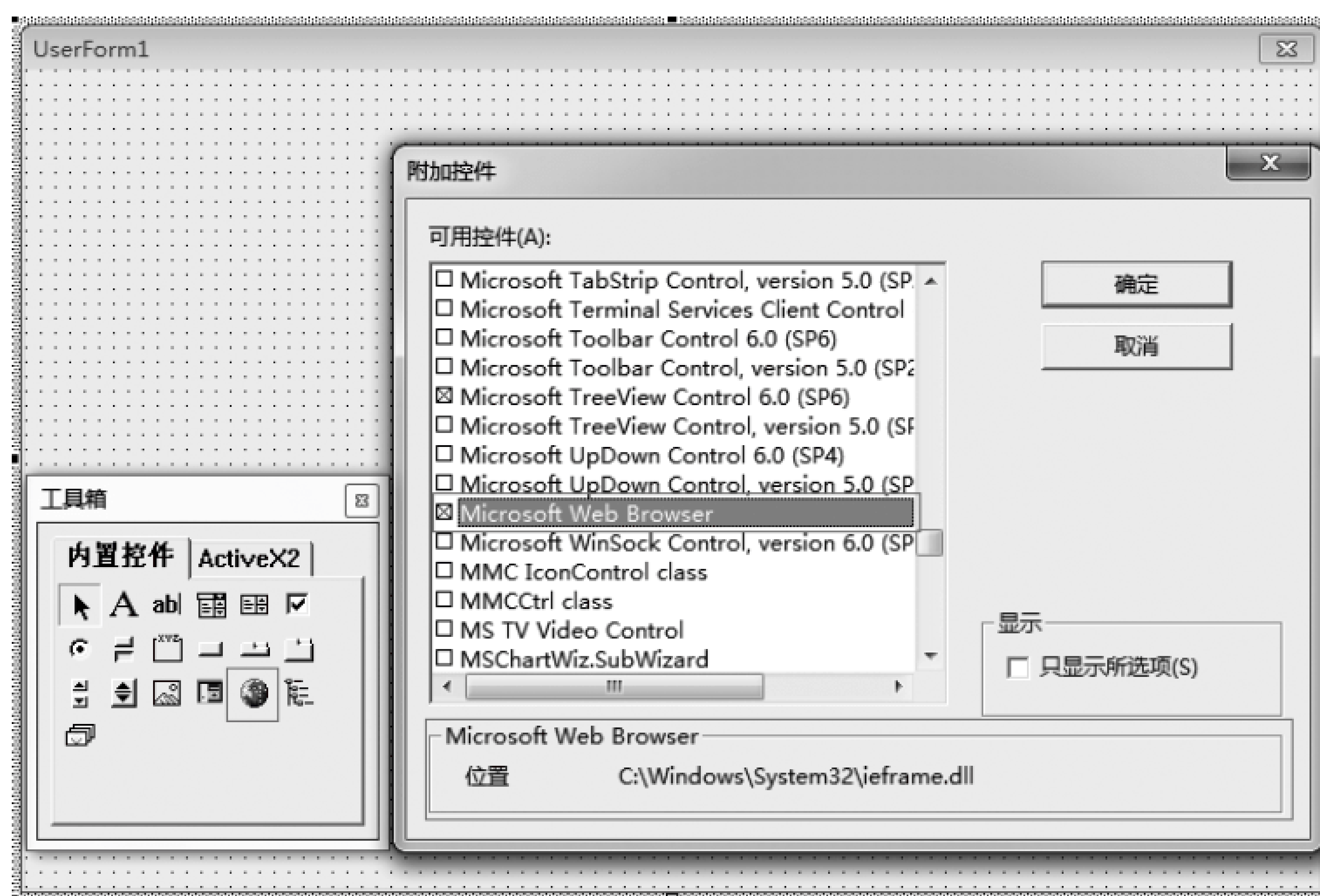


图 4-4　用户窗体使用 WebBrowser 控件

然后添加一个 TextBox 控件，用于设置 url，再放置一个命令按钮，并且命名为“显示”。命令按钮的单击事件如下。

```
Private Sub CommandButton1_Click()
    With Me.WebBrowser1
        .Navigate Me.TextBox1.Text
    End With
End Sub
```

写好代码后，启动窗体，在文本框中输入本地 XML 文件的路径，单击“显示”按钮，效果如图 4-5 所示。

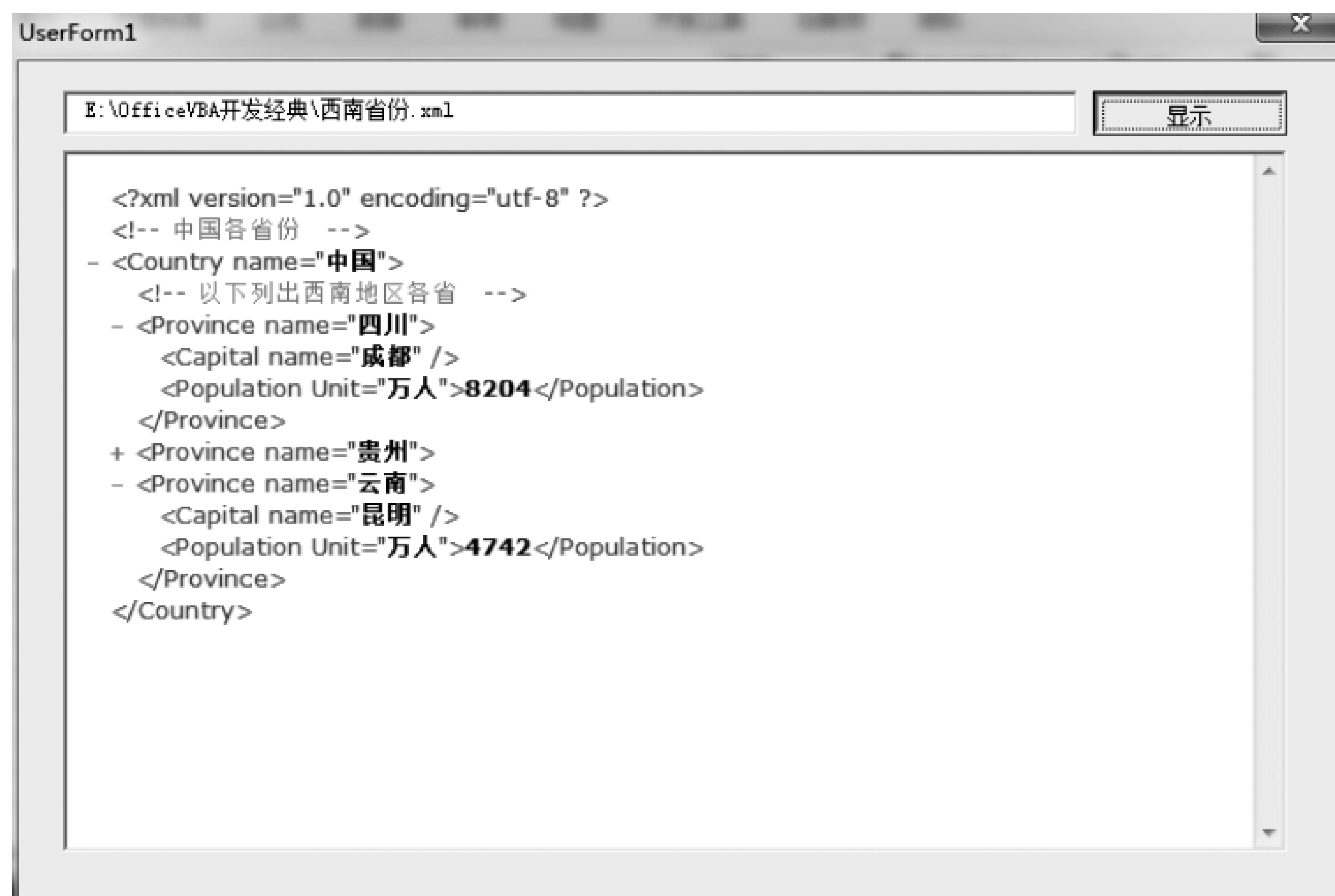


图 4-5　在 WebBrowser 控件中显示 XML

4.4　使用 DOMDocument 读写 XML

既然 XML 文件是文本文件，那么当然可以用文本文件读写的方式对 XML 文件进行存取。微软提供了专门针对 XML 的对象模型 DOM（Document Object Model），可以对 XML 进行全面操作。

4.4.1　引入 DOMDocument 对象

前期绑定：在 VBA 工程中添加引用“Microsoft XML, v 6.0”，如图 4-6 所示。

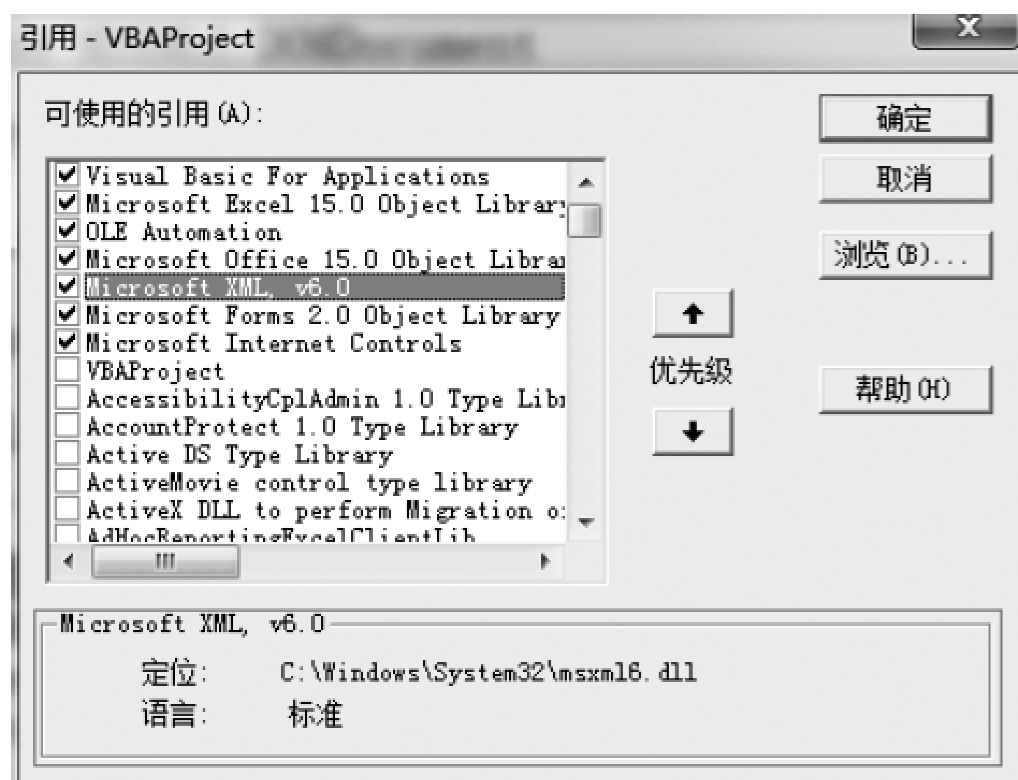


图 4-6　添加外部引用

添加该外部引用后，代码中使用

```
Dim Doc As MSXML2.DOMDocument
```

就声明了一个对象变量。

后期创建 DOMDocument 对象如下。

```
Set Doc = CreateObject("MSXML2.DOMDocument")
```

下面分别讲述把各种来源的 XML 载入 DOMDocument 的方法。

4.4.2　装载本地文件

使用 DOMDocument 对象的 Load 函数，可以把本地 XML 文件装载到 DOM 对象模型中。

语法如下。

```
Load(xmlSource)
```

参数 xmlSource 是指 XML 文件的路径，如果装载成功，Load 函数返回 True。

下面的过程装载本地 XML 文件，并在立即窗口打印文档对象的 XML 代码。

```
Sub OpenLocalXML()
    Dim Doc As MSXML2.DOMDocument
    Set Doc = New DOMDocument
```

```
    If Doc.Load(xmlSource:="E:\ 西南省份 .xml") Then
        Debug.Print Doc.XML
    End If
End Sub
```

代码分析：无论是 DOMDocument，还是其他的各种节点，几乎都有一个 XML 属性，该属性返回的是对象的 XML 代码字符串。因此可以通过调用该属性来查看节点的情况。

4.4.3 装载网络文件

很多情况下，XML 文件是网络上的一个远程文件，此时可以使用 XMLhttp 对象通过 URL 请求，返回一个 responseXML，然后赋给 DOMDocument 对象即可。

下面的代码从 w3school 网站上获取一个 XML 文件，然后装载到程序中的 DOMDocument 对象中。

```
Sub GetRemoteXML()
    Dim DOC As MSXML2.DOMDocument
    Dim X As MSXML2.XMLHTTP
    Set X = New XMLHTTP
    With X
        .Open "GET", "http://www.w3school.com.cn/example/xmle/simple.xml", False
        .send
        Set DOC = .responseXML
    End With
    Debug.Print DOC.XML
End Sub
```

运行上述代码，立即窗口打印出 XML 文件的源代码。

4.4.4 装载字符串

除了使用 Load 函数装载文件外，还可以使用 LoadXML 函数直接装载程序中的字符串。

下面的过程把程序中的字符串装载到 DOMDocument 中。

```
Sub LoadString()
    Dim DOC As MSXML2.DOMDocument
    Set DOC = New MSXML2.DOMDocument
    Dim code As String
    code = "<天气 date='2017-12-30'> <风力>2-3 级 </风力> <降水概率>60%</降水概率>
<最低气温>10℃ </最低气温> </天气>"
    If DOC.LoadXML(code) Then
        Debug.Print DOC.XML
    Else
        MsgBox "你的代码不合法，装载失败！", vbExclamation
    End If
End Sub
```

代码分析：如果变量 code 中的字符串不符合 XML 的语法规则，会弹出“你的代码不合法，装载失败！”的警告对话框。

上述过程运行后，立即窗口显示的运行结果如图 4-7 所示。

注意，创建 XML 的字符串中，表示元素的属性值时，可以用单引号代替双引号。

```
立即窗口
<天气 date="2017-12-30">
    <风力>2-3级</风力>
    <降水概率>60%</降水概率>
    <最低气温>10℃</最低气温>
</天气>
```

图 4-7　XML 文档装载字符串

4.4.5　保存 XML 文件

不管是从哪一个来源装载的 DOMDocument，随时都可以使用 DOMDocument 对象的 Save 方法保存为本地 XML 文件。用法如下。

```
DOC.Save "C:\build\temp.xml"
```

其中 DOC 就是一个 DOMDocument 对象，运行上述语句，会在指定路径下生成一个 XML 文件。

4.5　DOM 对象模型

XML 文件的 DOM 对象模型比较复杂，大致的对象模型如图 4-8 所示。

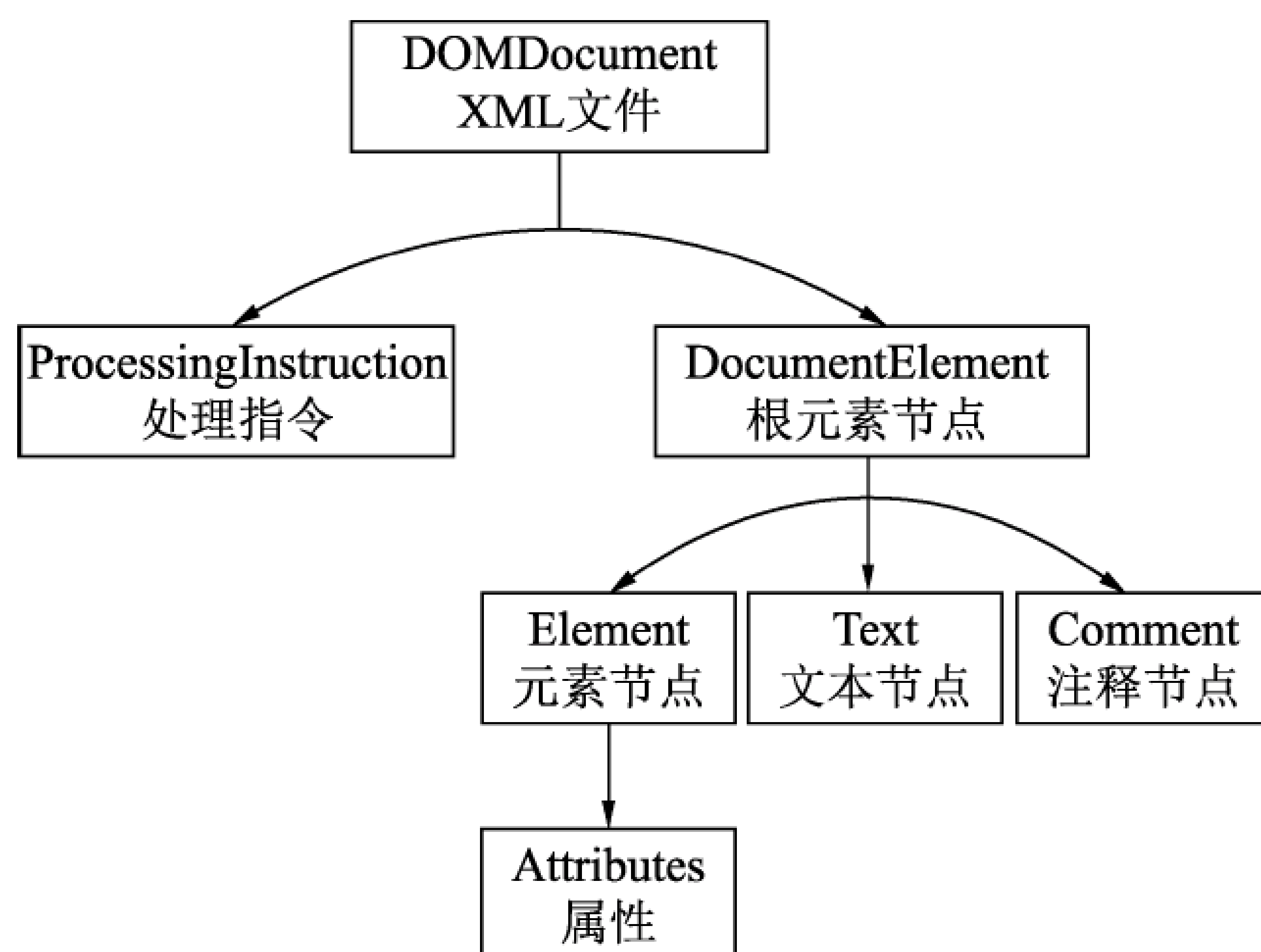


图 4-8　XML DOM 对象模型示意图

在图 4-8 中的任意一个方框都是一个节点，因此，DOMDocument 是 XML 文件的根节点，DocumentElement 是根元素节点。其实 DocumentElement 与其他元素节点没什么两样，不同的是这个节点的父节点是 DOMDocument。

因此，在 DOM 对象模型中，一切都围绕着“节点”来展开讨论的，一切对象皆为节点。

4.5.1　节点类型

XML 文件的 DOM 对象模型中的所有节点的通用类型是 MSXML2.IXMLDOMNode，也就是 DOM 节点对象，DOM 节点细分为十多种具体的节点类型（NodeType），如表 4-1 所示。

表 4-1 XML 节点类型

节 点 类 型	节点类型枚举常量	整 数 值
元素节点	NODE_ELEMENT	1
属性节点	NODE_ATTRIBUTE	2
文本节点	NODE_TEXT	3
CDATA 节点	NODE_CDATA_SECTION	4
实体引用名称节点	NODE_ENTITY_REFERENCE	5
实体名称节点	NODE_ENTITY	6
处理指令节点	NODE_PROCESSING_INSTRUCTION	7
注释节点	NODE_COMMENT	8
文档节点	NODE_DOCUMENT	9
文档类型节点	NODE_DOCUMENT_TYPE	10
文档片段节点	NODE_DOCUMENT_FRAGMENT	11
DTD 声明节点	NODE_NOTATION	12

4.5.2 节点对象

任意一个节点都可以声明为 MSXML2.IXMLDOMNode 对象，对应的集合对象是 MSXML2.IXMLDOMNodeList 对象，可以用来描述多个节点组成的集合。

细分到具体的节点类型，还可以用下面的对象类型来声明具体的节点，如表 4-2 所示。

表 4-2 XML 节点对象类型

节 点 类 型	对 象 类 型
文档节点	MSXML2.DOMDocument
元素节点	MSXML2.IXMLDOMElement
属性	MSXML2.IXMLDOMAttribute
文本节点	MSXML2.IXMLDOMText
注释节点	MSXML2.IXMLDOMComment
处理指令节点	MSXML2.IXMLDOMProcessingInstruction

4.5.3 节点对象的属性

要了解一个节点，一般要看它的 NodeName、NodeValue 和 NodeType 属性，这三个属性分别表示节点的名称、值和类型。

此外，还可以打印节点的 XML 属性，更全面地了解一个节点的信息。

4.6 定位节点

其实，学习 XML 的 DOM 对象模型，就是为了更方便地对节点进行定位、读取和修改等操作。由于 XML 是一个树状结构，节点之间有的是并列关系，有的是所属关系，准确定

位到某一节点，通常有很多种方法。以下介绍常用的定位方法。

4.6.1　使用 ChildNodes 定位所有子节点

XML 文件中的一个节点所包含的子节点用 ChildNodes 集合对象表示。xx.ChildeNodes(0) 就表示 xx 节点的首个子节点。

仍然以“西南省份 .xml”为例，该文档对象直属子节点有 3 个，即处理指令节点、注释节点和根元素节点。XML 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 中国各省份 -->
<Country name=" 中国 ">
    <!-- 以下列出西南地区各省 -->
    <Province name=" 四川 ">
        <Capital name=" 成都 "></Capital>
        <Population Unit=" 万人 ">8204</Population>
    </Province>
    <Province name=" 贵州 ">
        <Capital name=" 贵阳 "></Capital>
        <Population Unit=" 万人 ">3530</Population>
    </Province>
    <Province name=" 云南 ">
        <Capital name=" 昆明 "></Capital>
        <Population Unit=" 万人 ">4742</Population>
    </Province>
</Country>
```

下面的过程遍历文档对象的所有子节点（只包括直属子节点，不包括孙节点等）。

```
Sub LoopAllNodes()
    Dim DOC As MSXML2.DOMDocument
    Dim ND As MSXML2.IXMLDOMNode
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\ 西南省份.xml") Then
        Debug.Print " 节点名称 ", " 节点类型 ", " 节点值 "
        For Each ND In DOC.ChildNodes
            Debug.Print ND.nodeName, ND.NodeType, ND.NodeValue
        Next ND
    End If
End Sub
```

代码分析：由于各个子节点的类型不一样，因此代码中的对象变量 ND 需要声明为通用类型 .IXMLDOMNode，而不能是 IXMLDOMElement。

运行上述过程，在立即窗口打印各子节点的名称、类型和值，如图 4-9 所示。

```
立即窗口
节点名称      节点类型    节点值
xml           7           version="1.0" encoding="utf-8"
#comment      8           中国各省份
Country       1           Null
```

图 4-9　遍历子节点

需要注意的是，在 VBA 中 DOM 对象模型的集合都是以 0 作为第一个子成员的下标。此外，节点下面的 FirstChild 也可以定位到第一个子节点，它等价于 ChildNodes(0)，相对应地，LastChild 可以定位到最后一个子节点。

4.6.2 使用 PreviousSibling 和 NextSibling 定位前后节点

PreviousSibling 与 NextSibling 表示与节点并列的其他节点，分别表示前一个节点和后一个节点。

“西南省份 .xml”中的根元素节点是 <Country>，该节点下面包含 4 个子节点，分别是注释节点和 3 个元素节点。下列过程用于获取兄弟节点。

```
Sub 获取兄弟节点 ()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim ND(1 To 3) As MSXML2.IXMLDOMElement
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\西南省份.xml") Then
        Set Root = DOC.DocumentElement
        Debug.Print Root.ChildNodes.Length     '打印根元素节点的子节点总数
        Set ND(2) = Root.ChildNodes(2)
        Set ND(1) = ND(2).PreviousSibling
        Set ND(3) = ND(2).NextSibling
        Debug.Print ND(1).XML
        Debug.Print ND(3).XML
    End If
End Sub
```

代码分析：本例中 ND() 是一个数组，分别表示 3 个 Province，由于这 3 个都是元素节点，因此数组的类型可以声明为具体的 IXMLDOMElement。

上述程序的运行结果如图 4-10 所示。

```
立即窗口
 4
<Province name="四川">
    <Capital name="成都"></Capital>
    <Population Unit="万人">8204</Population>
</Province>
<Province name="云南">
    <Capital name="昆明"></Capital>
    <Population Unit="万人">4742</Population>
</Province>
```

图 4-10 获取兄弟节点

4.6.3 使用 ParentNode 定位父节点

与 ChildNodes 对应的是 ParentNode，ParentNode 是指当前节点的上一级节点。由于任一节点的父节点只能是一个，所以该单词后面没有 s。

仍然以“西南省份 .xml”为例。

在下面的过程中，首先定位到根元素下面的第 0 个子节点，这是一个注释节点。然后以

注释节点为基准，依次回溯其上级节点。

```
Sub 获取父级节点 ()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim ND  As MSXML2.IXMLDOMNode
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\ 西南省份 .xml") Then
        Set Root = DOC.DocumentElement
        Set ND = Root.FirstChild
        Debug.Print ND.nodeName, ND.ParentNode.nodeName, ND.ParentNode.ParentNode.
nodeName
    End If
End Sub
```

代码分析：ND 是一个注释节点，其上级是根元素节点 Country，再上一级就是文档对象。

运行上述过程，打印结果如下。

```
#comment      Country       #document
```

注意　如果从 XML 中的任一节点直接获取根元素节点（祖先节点），可以使用 ownerDocument，不需要反复使用 ParentNode。

4.6.4　使用 XPath 定位到任一节点

虽然在实际编程中经常使用 ChildNodes 定位子节点，但是如果一个 XML 的嵌套层数非常多，需要多次用到 ChildNodes，这时定位就显得麻烦了。

下面介绍使用 XPath 指定一个路径，从指定节点一步到达定位的深层节点。常用的节点定位的 XPath 写法如表 4-3 所示。

表 4-3　XPath 常用表示形式及其含义

XPath	含　义
/Country/Province/Capital	从根元素开始，逐级查找子节点。属于绝对路径
//Population	从当前节点开始，查找所有名称为 Population 的子节点
//Province/Capital[@name=' 昆明 ']	从当前节点开始，查找所有 Capital 子节点（name 属性必须是昆明）
//Province[Population<5000]	查找所有 Population 的文本小于 5000 的 Province 节点
//Province[Population<5000]/Capital	查找所有 Population 的文本小于 5000 的 Province 下面的 Capital 子节点
/Country/*	查找 Country 之下的所有元素节点（不包括其他类型节点）
/Country/Province/@*	查找 Province 的所有属性节点
/Country/node()	查找 Country 之下的所有类型的子节点

更多 XPath 的高级用法，请参阅其他资料。

1. 使用 SelectSingleNode 获取第一个符合路径的节点

下面的过程使用绝对路径的方式查找到第一个名称为 Capital 的元素节点。

```
Sub XPath1()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim ND  As MSXML2.IXMLDOMNode
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\ 西南省份.xml") Then
        Set Root = DOC.DocumentElement
        Set ND = Root.SelectSingleNode("/Country/Province/Capital")
        Debug.Print ND.XML
    End If
End Sub
```

代码分析：/Country/Province/Capital 是一个 XPath，表示从根元素开始，逐级定位，一直查到 Capital 为止。虽然本例中的 XML 文件中有 3 个 Capital 节点，但是程序中用的是 SelectSingleNode 方法，因此找到第一个即可。

运行上述过程，打印结果如下。

```
<Capital name=" 成都 "></Capital>
```

2. 使用 SelectNodes 获取所有符合路径的节点集合

SelectNodes 方法返回的是多个节点构成的集合，因此需要声明为 IXMLDOMNodeList 类型。

```
Sub XPath2()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim NL  As MSXML2.IXMLDOMNodeList, ND As MSXML2.IXMLDOMNode
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\ 西南省份.xml") Then
        Set Root = DOC.DocumentElement
        Set NL = Root.SelectNodes("/Country/Province/Capital")
        For Each ND In NL
            Debug.Print ND.XML
        Next ND
    End If
End Sub
```

代码分析：对象变量 NL 获取了多个符合该路径的节点，因此还需要用 For 循环遍历每个子节点的信息。

运行上述过程，打印结果如下。

```
<Capital name=" 成都 "></Capital>
<Capital name=" 贵阳 "></Capital>
<Capital name=" 昆明 "></Capital>
```

4.6.5 使用 getElementsByTagName 定位到一组元素节点

getElementsByTagName 方法是以当前节点为基准，查找指定名称的所有元素节点。

下面的过程从文档对象开始查找所有 Population 元素节点。

```
Sub GetByTagName()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim NL As MSXML2.IXMLDOMNodeList, ND As MSXML2.IXMLDOMNode
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\西南省份.xml") Then
        Set Root = DOC.DocumentElement
        Set NL = DOC.getElementsByTagName("Population")
        For Each ND In NL
            Debug.Print ND.XML
        Next ND
    End If
End Sub
```

代码分析：这个实例的功能等价于 XPath 中的 //Population。

运行上述过程，打印结果如下。

```
<Population Unit="万人">8204</Population>
<Population Unit="万人">3530</Population>
<Population Unit="万人">4742</Population>
```

4.6.6　使用 getAttributeNode 定位到属性

元素节点的 getAttributeNode 方法可以定位到元素中的某一属性，返回一个属性节点。

```
Sub GetByAttributeNode()
    Dim DOC As MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim A As MSXML2.IXMLDOMAttribute
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\西南省份.xml") Then
        Set Root = DOC.DocumentElement
        Set A = Root.getAttributeNode("name")
        Debug.Print A.nodeName, A.NodeValue, A.NodeType
    End If
End Sub
```

代码分析：代码中的 Root 是根元素节点（Country），对象变量 A 是一个属性节点，表示 Country 元素的 name 属性。

运行上述过程，打印结果如下。

```
name          中国          2
```

注意　属性一般书写在元素节点的开始标签，但是每一个属性并非元素节点的子节点，因此元素与属性之间的关系不能通过 ChildNodes 或者 ParentNode 来描述。

以上内容的源代码文件为“实例文档 12.xlsm”。

4.7　详细了解元素节点

元素节点是构成 XML 树状结构的骨架，可以说文本节点、注释节点、属性等都附着于

元素节点。因此，从一个元素节点出发，获取与该元素有关的其他节点、属性具有非常重要的意义。

一个元素节点往往具有它的父子、兄弟节点，这些前面已经介绍过，本节重点讲述元素节点下面包含的属性、子元素、文本节点和注释节点。

本节案例一律基于“华北地区 .xml”文件，如图 4-11 所示。

```
<Country name="中国">
  <Area name="华北地区">
    <Province name="河北">
        <Capital name="石家庄"/>
      </Province>
    <Province name="内蒙古" eng="Inner Mongolia" shortname="蒙">
        青城
        <City name="呼和浩特"/>
      <!--区号: 0471 -->
        稀土之都
        <City name="包头"/>
      <!--区号: 0472 -->
        塞外小北京
        <City name="巴彦浩特"/>
      <!--区号: 0483 -->
      </Province>
    <Province name="山西">
        <Capital name="太原"/>
      </Province>
    </Area>
 </Country>
```

图 4-11 “华北地区 .xml”文件内容

首先熟悉一下文件结构，第一个 Province 元素名称是“内蒙古”，该节点的父级节点是 Area，祖先节点是 Country。

Province 元素节点包含 3 个属性、3 个文本节点、3 个子元素节点（City）、3 个注释节点。

4.7.1 遍历元素的属性

元素节点的属性，使用 Attributes 集合对象描述。用 For 循环可以遍历每一个属性的名称和值。

下面的过程首先用 XPath 定位到内蒙古这个 Province 元素节点，然后打印该节点的所有属性。

```
Sub GetAllAttributes()
    Dim DOC As MSXML2.DOMDocument
    Dim province As MSXML2.IXMLDOMElement
    Dim attr As MSXML2.IXMLDOMAttribute
    Set DOC = New DOMDocument
    If DOC.Load(xmlSource:="E:\华北地区.xml") Then
        Set province = DOC.SelectSingleNode("/Country/Area/Province[@name='内蒙古']")
        For Each attr In province.Attributes
            Debug.Print attr.Name, attr.Value
        Next attr
```

```
        Debug.Print " 下面只获取 eng 属性 "
        Debug.Print province.getAttribute("eng")
    End If
End Sub
```

代码分析：由于需要从 Attributes 集合中遍历，因此循环变量 attr 需要声明为 IXMLDOMAttribute 类型。

"/Country/Area/Province[@name=' 内蒙古 ']" 表示用绝对路径找到 Province 元素节点，并且 name 属性必须等于“内蒙古”。另外，XPath 也可以写作 "/Country/Area/Province[1]"，表示找到的所有 Province 节点中只选取第 1 个，也就是内蒙古的那个 Province。

如果要单独获取某一属性，可以使用节点的 getAttribute 函数。

运行上述过程，立即窗口的结果如图 4-12 所示。

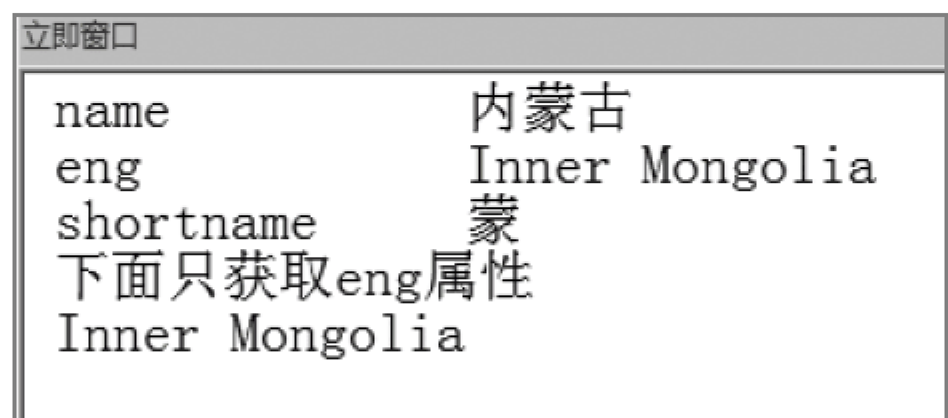


图 4-12　遍历元素的属性

4.7.2　遍历元素的文本节点

元素节点下面经常包含多种不同类型的子节点，因此在 ChildNodes 里面遍历时，需要用通用的节点类型 IXMLDOMNode，如果只遍历文本节点，还需要使用 NodeType 或 NodeTypeString 判断一下。

下面的过程遍历 Province 的 3 个文本节点。

```
Sub GetAllTextNodes()
    Dim DOC As MSXML2.DOMDocument
    Dim province As MSXML2.IXMLDOMElement
    Dim node As MSXML2.IXMLDOMNode
    Dim T As MSXML2.IXMLDOMText
    Set DOC = New DOMDocument
    DOC.Load xmlSource:="E:\ 华北地区 .xml"
    Set province = DOC.SelectSingleNode("/Country/Area/Province[@name=' 内蒙古 ']")
    For Each node In province.ChildNodes
        If node.NodeType = NODE_TEXT Then
            Set T = node
            Debug.Print T.nodeName, T.nodeTypeString, T.NodeValue
        End If
    Next node
End Sub
```

代码分析：NodeType 是一个枚举常量值（整数值），与其对应的是 NodeTypeString（字符串），这两个都可以用来判断节点的类型。

运行上述过程，立即窗口的结果如图 4-13 所示。

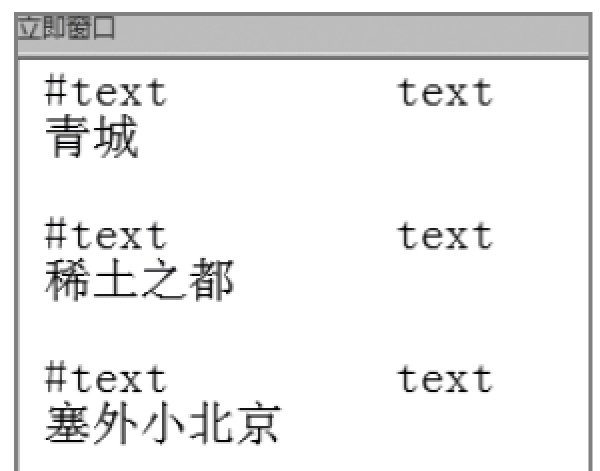


图 4-13　遍历文本节点

4.7.3 遍历元素的子元素节点

遍历子元素节点时，只需要判断 NodeType 是否等于 NODE_ELEMENT，或者 NodeTypeString 是否等于“element”即可。

下面的过程判断 Province 下面的 3 个 City 子元素节点。

```
Sub GetAllElements()
    Dim DOC As MSXML2.DOMDocument
    Dim province As MSXML2.IXMLDOMElement
    Dim node As MSXML2.IXMLDOMNode
    Dim E As MSXML2.IXMLDOMElement
    Set DOC = New DOMDocument
    DOC.Load xmlSource:="E:\华北地区.xml"
    Set province = DOC.SelectSingleNode("/Country/Area/Province[@name='内蒙古']")
    For Each node In province.ChildNodes
        If node.NodeType = NODE_ELEMENT Then
            Set E = node
            Debug.Print E.nodeName, E.nodeTypeString, E.XML
        End If
    Next node
End Sub
```

运行上述过程，立即窗口的结果如图 4-14 所示。

```
立即窗口
City          element       <City name="呼和浩特"/>
City          element       <City name="包头"/>
City          element       <City name="巴彦浩特"/>
```

图 4-14　遍历子元素节点

4.7.4 遍历元素的注释节点

针对 IXMLDOMElement 对象，nodeName 的属性是 #comment，通过访问 data 属性可以获得注释符号里面的内容。

```
Sub GetAllComments()
    Dim DOC As MSXML2.DOMDocument
    Dim province As MSXML2.IXMLDOMElement
    Dim node As MSXML2.IXMLDOMNode
    Dim C As MSXML2.IXMLDOMComment
    Set DOC = New DOMDocument
    DOC.Load xmlSource:="E:\华北地区.xml"
    Set province = DOC.SelectSingleNode("/Country/Area/Province[@name='内蒙古']")
    For Each node In province.ChildNodes
        If node.NodeType = NODE_COMMENT Then
            Set C = node
            Debug.Print C.nodeName, C.nodeTypeString, C.Data
        End If
    Next node
End Sub
```

运行上述过程，立即窗口的结果如图 4-15 所示。

```
立即窗口
#comment        comment        区号：0471
#comment        comment        区号：0472
#comment        comment        区号：0483
```

图 4-15　遍历注释节点

4.8　创建和修改 XML

前面讲解的都是基于现有的 XML 文件来读取内容信息。在实际编程应用中，经常需要对现有 XML 中的部分内容进行修改，或者完全从头创建一个 XML。

4.8.1　创建节点

XML 文件中的一切都是节点，只不过各个节点类型不同而已。DOMDocument 对象下面有一些以 create 开头的方法，这些方法用于创建各种类型的节点。

- createAttribute(name)：创建一个指定属性名称的属性。
- createComment(data)：创建一个注释，注释的内容为 data。
- createElement(tagName)：用于创建一个指定名称的元素节点。
- createNode(type,name,namespaceURI)：创建一个指定节点类型和节点名称的节点。
- createProcessingInstruction(target,data)：创建一个处理指令。
- createTextNode(data)：创建一个文本节点，内容为 data。

其中，createAttribute 方法返回一个属性对象，该属性对象不能用 AppendChild 等方法插入元素中，因为属性不是元素节点的子节点。

下面的过程用于分别创建以上 6 种类型的节点。

```
Sub 创建节点 ()
    Dim DOC As MSXML2.DOMDocument                  ' 声明文档对象
    Dim attr As IXMLDOMAttribute                   ' 声明一个属性变量
    Dim comment As IXMLDOMComment                  ' 声明一个注释变量
    Dim element As IXMLDOMElement                  ' 声明一个元素变量
    Dim node As IXMLDOMNode                        ' 声明一个通用节点
    Dim instruction As IXMLDOMProcessingInstruction ' 声明一个处理指令
    Dim text As IXMLDOMText                        ' 声明一个文本节点
    Set DOC = New MSXML2.DOMDocument               ' 新建一个 XML 文档
    With DOC
        Set attr = .createAttribute("age")         ' 创建一个属性节点
        attr.Value = "36"                          ' 设置属性值
        Debug.Print attr.XML

        Set comment = .createComment(" 去年的信息 ")  ' 创建一个注释节点
        Debug.Print comment.XML

        Set element = .createElement("staff")  ' 创建一个名称为 staff 的元素节点
        element.setAttribute "name", "yongfu liu"
        element.setAttribute "gender", "male"
        Debug.Print element.XML

        Set node = .createNode(Type:=MSXML2.NODE_ELEMENT, Name:="manager",
```

```
NamespaceURI:="")
            Debug.Print node.XML

            Set instruction = .createProcessingInstruction("xml", " version='1.0'
encoding='utf-8'")
            Debug.Print instruction.XML

            Set text = .createTextNode("This is a message")
            Debug.Print text.XML
        End With
    End Sub
```

代码分析：每创建一个节点，就打印该节点的代码。但是由于这些节点均未附加到 DOC 中，因此未形成一个完整的 XML 节点树。

运行上述过程，立即窗口打印出各个节点的 XML 代码，如图 4-16 所示。

```
立即窗口
age="36"
<!--去年的信息-->
<staff name="yongfu liu" gender="male"/>
<manager/>
<?xml version="1.0"?>
This is a message
```

图 4-16　创建各种类型的节点

4.8.2　插入节点

创建了的节点尚未出现在 XML 文档中，必须通过 AppendChild 方法或 InsertBefore 方法把节点作为 XML 文档已有节点的子节点插入。

例如 A.AppendChild B，表示把 B 节点成为 A 节点的子节点，如果 A 中已经有子节点，那么 B 被放到最后位置。

InsertBefore 方法与 AppendChild 类似，不同的是该方法可以指定插入的位置。例如：A.InsertBefore B,C 表示 C 已经是 A 的一个子节点了，接下来要把 B 也作为 A 的子节点，但是要放在 C 之前。

节点之间通过反复使用 AppendChild 这些方法，就把零散的节点形成了节点树。

下面的过程把创建的各个节点组织成一个完整的 XML 文件。

```
    Sub 插入节点 ()
        Dim DOC As MSXML2.DOMDocument                         ' 声明文档对象
        Dim attr As IXMLDOMAttribute                          ' 声明一个属性变量
        Dim comment As IXMLDOMComment                         ' 声明一个注释变量
        Dim element As IXMLDOMElement                         ' 声明一个元素变量
        Dim node As IXMLDOMNode                               ' 声明一个通用节点
        Dim instruction As IXMLDOMProcessingInstruction       ' 声明一个处理指令
        Dim text As IXMLDOMText                               ' 声明一个文本节点
        Set DOC = New MSXML2.DOMDocument                      ' 新建一个 XML 文档
        With DOC
            Set attr = .createAttribute("age")                ' 创建一个属性节点
            attr.Value = "36"                                 ' 设置属性值
            Set comment = .createComment(" 去年的信息 ")        ' 创建一个注释节点
            Set element = .createElement("staff")   ' 创建一个名称为 staff 的元素节点
            element.setAttribute "name", "yongfu liu"
            element.setAttribute "gender", "male"
            Set node = .createNode(Type:=MSXML2.NODE_ELEMENT, Name:="manager",
NamespaceURI:="")
```

```
            Set instruction = .createProcessingInstruction("xml", " version='1.0'
encoding='utf-8'")
            Set text = .createTextNode("This is a message")
            '以上创建节点完毕

            element.setAttributeNode attr    '把 age 属性添加到 staff 元素节点中
            element.appendChild node         '把 manager 元素节点作为 staff 元素的子节点
            element.appendChild comment      '把注释节点作为 staff 元素的子节点
            element.InsertBefore text, comment
                             '把文本节点作为 staff 元素的子节点，但是插入到注释节点之前
            DOC.appendChild element 'staff 元素作为文档对象的子节点，也就是作为根元素节点
            DOC.InsertBefore instruction, DOC.FirstChild
                             '把处理指令插入到 DOC 首个子节点之前，也就是处理指令置顶
            '以上插入节点完毕
            Debug.Print DOC.XML
            DOC.Save ThisWorkbook.Path & "\temp.xml"
        End With
    End Sub
```

运行上述过程，使用浏览器阅览 XML 文件，效果如图 4-17 所示。

可以看出，文本节点出现在注释节点上方，这体现了 InsertBefore 的作用。

```
<?xml version="1.0" ?>
- <staff name="yongfu liu" gender="male" age="36">
    <manager />
    This is a message
    <!-- 去年的信息 -->
  </staff>
```

图 4-17　插入节点

4.8.3　移除节点

移除节点与插入节点恰恰相反，是从节点中移除一个子节点，可以使用 RemoveChild 方法。例如：

A.RemoveChild A.FirstChild 表示把 A 的第 0 个子节点移除。

A.RemoveChild A.ChildNodes(1) 表示把 A 的第 1 个子节点移除。

4.8.4　修改和移除节点的属性

添加、修改元素节点的属性，使用 setAttribute 方法；移除已有的属性，使用 removeAttribute 方法。

```
Sub 修改和移除元素节点的属性()
    Dim DOC As MSXML2.DOMDocument
    Dim Elem As MSXML2.IXMLDOMElement
    Set DOC = New DOMDocument
    Set Elem = DOC.createElement("Staff")        '创建一个元素节点
    Elem.setAttribute "name", "John"             '添加 name 属性
    Elem.setAttribute "hobby", "Music"           '添加 hobby 属性
    Elem.setAttribute "age", "25"                '添加 age 属性
    Elem.setAttribute "hobby", "Traveling"       '修改 hobby 属性
    Elem.removeAttribute "age"                   '移除 age 属性
    Debug.Print Elem.XML
End Sub
```

运行上述过程，立即窗口的输出结果如下。

```
<Staff name="John" hobby="Traveling"/>
```

4.8.5 替换节点

使用 replaceChild 方法可以把节点中已有的一个子节点用新的节点替换掉。语法如下。

```
A.replaceChild newChild,oldChild
```

其中，A 是一个节点，oldChild 是 A 节点已有的一个子节点，newChild 是其他节点。

下面的过程在名称为 Staff 的元素节点下面添加一个注释节点，然后用新建的 ContactInformation 元素节点替换注释节点。

```
Sub 替换节点 ()
    Dim DOC As MSXML2.DOMDocument
    Dim Elem As MSXML2.IXMLDOMElement
    Dim nd1 As MSXML2.IXMLDOMComment, nd2 As MSXML2.IXMLDOMElement
    Set DOC = New DOMDocument
    Set Elem = DOC.createElement("Staff")       '创建一个元素节点
    Elem.setAttribute "name", "John"
    Set nd1 = DOC.createComment("This is a comment")
    Elem.appendChild nd1
    Set nd2 = DOC.createElement("ContactInformation")
    nd2.setAttribute "phone", "13612345678"
    nd2.setAttribute "address", "Beijing"
    Elem.replaceChild newChild:=nd2, oldChild:=nd1
    DOC.appendChild Elem
    Debug.Print DOC.XML
    DOC.Save ThisWorkbook.Path & "\temp.xml"
End Sub
```

上述过程运行后，浏览器中看到的 temp.xml 文件结果如图 4-18 所示。

```
- <Staff name="John">
    <ContactInformation phone="13612345678" address="Beijing" />
  </Staff>
```

图 4-18 替换节点

4.8.6 克隆节点

CloneNode 方法可以把已有的节点复制一份，复制出来的节点可以插入其他节点中。克隆节点的语法如下。

```
Set C = B.Clone(deep)
```

其中，B 是 XML 中任意一个节点，C 是由 B 克隆出来的新节点，如果 deep 参数设置为 True，表示深层克隆，也就是 B 节点包含的深层子节点一同克隆；deep 为 False 时，只克隆 B 节点的外壳。

“西南省份 .xml”共有 3 个 Province 元素节点。下面的过程把名称为四川的 Province 元素节点克隆两份，一份采用表层克隆，另一份采用深层克隆。最后把克隆出来的两个节点插入 Country 根元素节点。

```
Sub 克隆节点 ()
    Dim DOC As MSXML2.DOMDocument
    Dim Elem As MSXML2.IXMLDOMElement
    Dim province As MSXML2.IXMLDOMElement
    Dim nd1 As MSXML2.IXMLDOMElement, nd2 As MSXML2.IXMLDOMElement
    Set DOC = New DOMDocument
    DOC.Load xmlSource:="E:\ 西南省份.xml"
    Set province = DOC.SelectSingleNode("/
Country/Province[@name=' 四川 ']")
    Set nd1 = province.CloneNode(deep:=False)
    Set nd2 = province.CloneNode(deep:=True)
    province.ParentNode.appendChild nd1
    province.ParentNode.appendChild nd2
    Debug.Print DOC.XML
    DOC.Save ThisWorkbook.Path & "\temp.xml"
End Sub
```

运行上述过程，产生的新文件 temp.xml 在浏览器中的效果如图 4-19 所示。

可以看出 nd1 节点只是原节点的表层，而 nd2 和原节点一模一样。

以上内容的源代码文件为“实例文档 13.xlsm”。

```
<?xml version="1.0" ?>
<!-- 中国各省份 -->
- <Country name="中国">
    <!-- 以下列出西南地区各省 -->
  - <Province name="四川">
      <Capital name="成都" />
      <Population Unit="万人">8204</Population>
    </Province>
  - <Province name="贵州">
      <Capital name="贵阳" />
      <Population Unit="万人">3530</Population>
    </Province>
  - <Province name="云南">
      <Capital name="昆明" />
      <Population Unit="万人">4742</Population>
    </Province>
    <Province name="四川" />   nd1
  - <Province name="四川">     nd2
      <Capital name="成都" />
      <Population Unit="万人">8204</Population>
    </Province>
</Country>
```

图 4-19　克隆节点

4.9　使用 Schema 验证 XML

通过 DOMDocument 对象的 Load 或 LoadXML 方法可以装载一个 XML 文件或字符串，如果 XML 文件符合语法基本规则，则认为是形式良好的（Well-Formed），Load 方法相应返回 True，表示装载成功。

但是在很多情况下，一个 XML 文件除了具有良好的形式外，还需要从内容的角度去判断是否满足某一个规则或格式，如果不满足内容要求，相应地给出原因。

对现有 XML 文件进行验证操作，需要了解和准备以下三方面内容。

- ❑ XSD 文件：用于建立规则或格式，该文件是 Schema 验证能否通过的核心文件。可以用记事本程序创建或使用专业工具，虽然扩展名是 .xsd，实质上也是一种 XML 文件。
- ❑ XMLSchemaCache 对象：在 VBA 程序中声明和创建，具体功能是连接 XSD 文件与 XML 文件。
- ❑ parseError 对象：解析错误对象，返回解析和验证的结果。

4.9.1　在 XSD 文件中创建规则

创建用于 Schema 验证的 XSD 文件，体系相当庞大。限于本书篇幅，此处仅以实例演示一下 XSD 与 XML 文件的呼应关系。

假设每个员工制作体现自己信息的一个 XML 文件，具体要求如下。

- 根元素节点名称：Resume。
- 根元素下面必须有 Name、Gender、Birthday、Phone 这些子元素。除了 Phone 是可选元素外，其余 3 个只能出现一次。
- Name 元素的文本内容是 2~3 个字符。
- Gender 只能选择 Male 或 Female。
- BirthDay 的格式必须是 YYYY-MM-DD。
- Phone 的格式必须是 11 位连续数字。

如图 4-20 所示是员工李四制作的 XML 文件“李四 .xml”，可以看出该文件完全符合上述要求。

```
 <?xml version="1.0" ?>
- <Resume xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <Name>李四</Name>
    <Gender>Female</Gender>
    <Birthday>2010-11-12</Birthday>
    <Phone>13612345678</Phone>
  </Resume>
```

图 4-20 “李四 .xml”文件内容

那么，哪些 XML 不符合要求呢，如何快速判断？这就需要创建 XSD 文件，如图 4-21 所示是笔者用记事本程序创建的用于验证上述 XML 的 XSD 文件，文件名为 ResumeTemplate.xsd。

```
 <?xml version="1.0" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://sch
    targetNamespace="http://schemas.microsoft.com/office/2009/07/customui"
  - <xs:element name="Resume">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="Name" minOccurs="1">
          - <xs:simpleType>
            - <xs:restriction base="xs:string">
                <xs:minLength value="2" />
                <xs:maxLength value="3" />
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        - <xs:element name="Gender" minOccurs="1">
          - <xs:simpleType>
            - <xs:restriction base="xs:string">
                <xs:enumeration value="Male" />
                <xs:enumeration value="Female" />
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        - <xs:element name="Birthday" minOccurs="1">
          - <xs:simpleType>
            - <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{4}-[0-9]{2}-[0-9]{2}" />
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        - <xs:element name="Phone" minOccurs="0">
          - <xs:simpleType>
            - <xs:restriction base="xs:integer">
                <xs:totalDigits value="11" />
              </xs:restriction>
            </xs:simpleType>
```

图 4-21 XML 验证文件

XSD 文件看起来错综复杂，但是大多数代码都是一成不变的。这里重点关注 ComplexType 和 SimpleType。ComplexType 表示是复杂元素，该元素下面可以包含子节点，而 SimpleType 相反，是简单节点，不能包含子节点。

注意图 4-21 中粗体字的部分，可以看出，Resume 元素是一个 ComplexType，它下面包含着 4 个 SimpleType。每一个 SimpleType 包含着 restriction（对元素的限制条件）。该文件用到的约束技巧如下。

- minOccurs：最少出现次数，0 表示可以不出现，1 表示必须出现一次。
- minLength、maxLength：最短字符数和最长字符数。
- enumeration：枚举值。
- pattern：用正则表达式约束输入格式。
- totalDigits：总的数字位数。

4.9.2　配置 DOMDocument 的 Schema

准备好 XSD 验证文件、XML 范例文件，就可以在 VBA 中进行验证了。验证的具体步骤如下。

（1）在 VBA 中创建一个 XMLSchemaCache60 变量 schema。

（2）为 schema 指定命名空间、XSD 文件路径。

（3）创建一个 DOMDocument 变量。

（4）用 DOMDocument 装载 XML 文件，装载的同时进行验证。

（5）查看 ErrorCode 是否为 0，如果不为 0 则调查错误原因。

假设员工张三创建的一个 XML 文件“张三 .xml”如图 4-22 所示。

```
<?xml version="1.0" ?>
- <Resume xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <Name>ZhangSan</Name>
    <Gender>女</Gender>
    <birthday>2010年11月12日</birthday>
    <Phone>136123456**</Phone>
  </Resume>
```

图 4-22　“张三 .xml”文件内容

下面的程序用于验证“张三 .xml”是否合法。

```
Sub SchemaValidate()
    Dim schema As MSXML2.XMLSchemaCache60         ' 声明一个 schema 验证变量
    Set schema = New XMLSchemaCache60             ' 创建一个 schema
    schema.Add NamespaceURI:="http://schemas.microsoft.com/office/2009/07/customui",
Var:=ThisWorkbook.Path & "\ResumeTemplate.xsd"
    ' 为 schema 变量添加命名空间，并且指定 XSD 文件的所在路径
    Dim DOC As New MSXML2.DOMDocument60
    With DOC
        Set .Schemas = schema                     ' 把 schema 变量与 DOM 文档关联
        .async = False
        .validateOnParse = True                   ' 解析时立即验证
```

```
            .resolveExternals = False
            .Load (ThisWorkbook.Path & "\ 张三 .xml") ' 装载计划验证的实例 XML 文件
                With .parseError                       ' 解析错误对象
                    If .ErrorCode = 0 Then
                        MsgBox " 通过 schema 验证。", vbInformation
                    Else
                        MsgBox " 错误原因：" & .reason & vbNewLine & " 引起错误的节点：" 
& .srcText, vbCritical
                    End If
                End With
        End With
    End Sub
```

代码分析：schema 变量、XSD 文件、XML 文件三者的命名空间必须是一致的，都为 http://schemas.microsoft.com/office/2009/07/customui，如果不一致，则达不到验证的效果。

运行上述过程，弹出如图 4-23 所示的错误对话框。

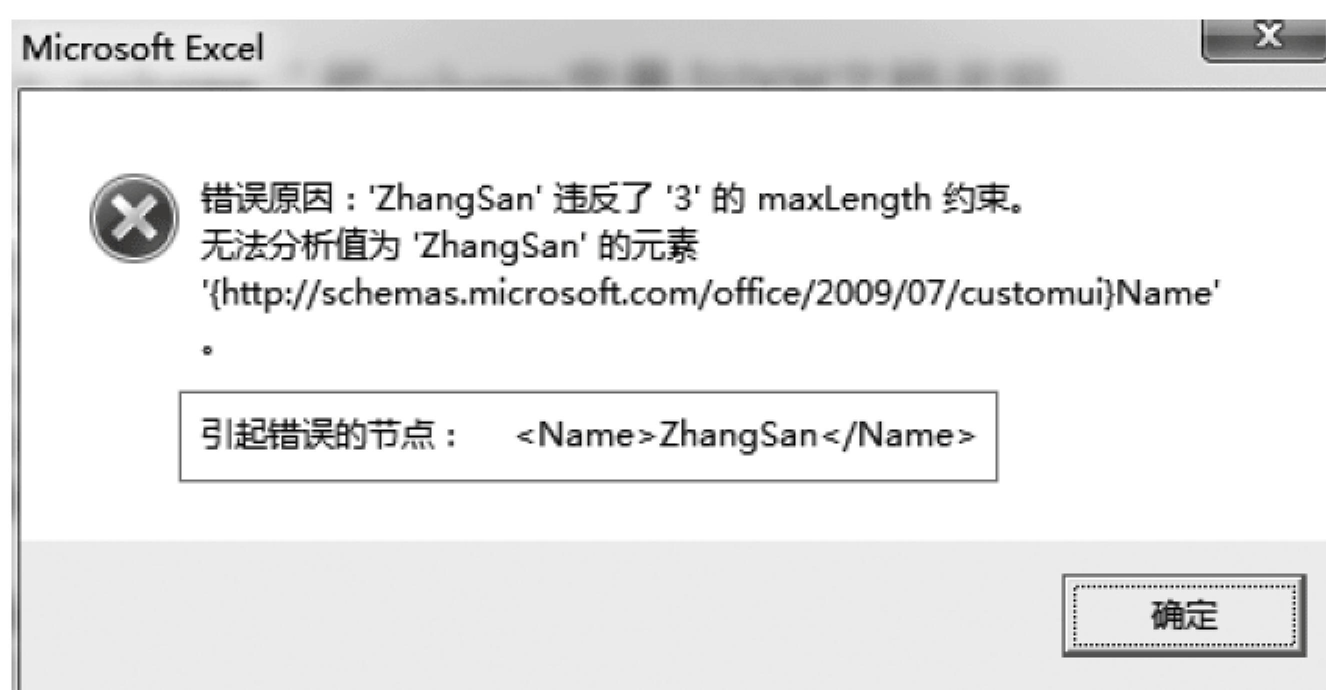


图 4-23　使用 XSD 文件验证 XML 时弹出错误

可以看出，错误原因是 Name 元素的文本太长。XSD 文件要求 2 ～ 3 个字符，实际是 8 个字符。

需要注意的是，如果 XML 文档存在多处错误，验证操作进行时只挑出第一个错误的信息，只有修复了第一个错误并再次验证，才能继续弹出后续的错误。

4.9.3　分析验证结果

parseError 是 DOMDocument 对象的一个成员，该成员包含如表 4-4 所示的属性，用来返回错误的各种信息。

表 4-4　XML 验证错误对象的成员

属　　性	描　　述
errorCode	返回一个长整型错误码
reason	返回包含错误原因的字符串
line	返回表示错误行号的长整型
linepos	返回表示错误的行位置的长整型
srcText	返回包含引起错误的行的字符串

续表

属　　性	描　　述
url	返回指向被加载文档的 URL
filepos	返回错误的一个长整型文件位置

其中，比较有用的属性有 errorCode、reason、strText、line 等。

以上内容的源代码文件为“实例文档 14.xlsm”。

4.10　XML 与 Office 文档

Office 文档（Excel 工作簿、Word 文档、PowerPoint 演示文稿）对象都有一个 CustomXMLParts 的集合对象。一般的文档默认有 3 个内置的 CustomXMLPart。

通过 VBA 可以为文档添加自定义 XML，也可以从文档中移除自定义 XML。无论是添加 XML 还是移除 XML，肉眼看不出来，需要用压缩软件打开才能看到变化。

4.10.1　添加自定义 XML 到 Word 文档

下面的实例在 Excel VBA 中编程，向处于打开状态的 Word 文档中添加自定义 XML。

首先为 Excel VBA 工程添加对 Word 的外部引用，如图 4-24 所示。

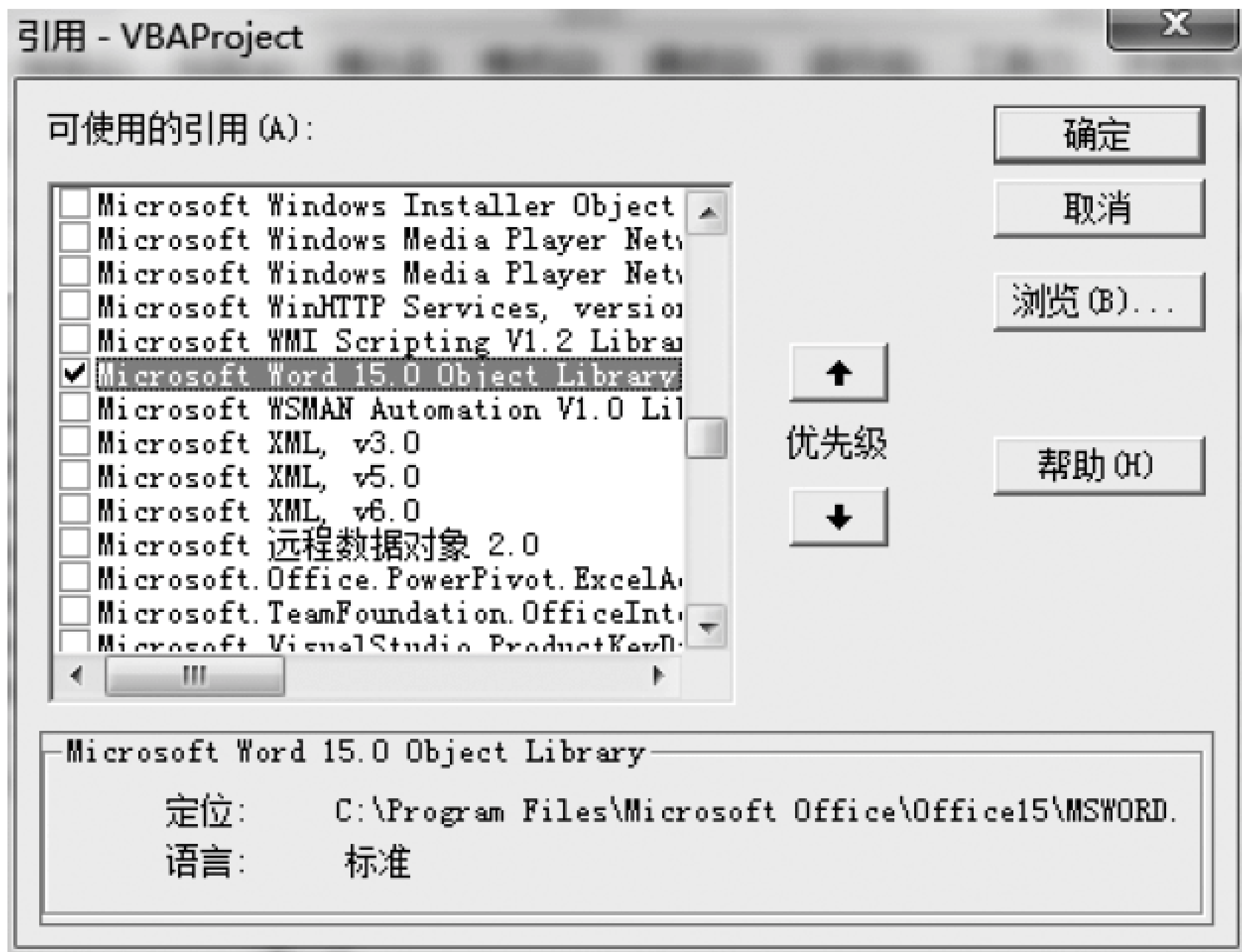


图 4-24　添加外部引用

然后运行下面的过程，向指定 Word 文档添加 XML，并且保存文档。

```
Sub 添加并遍历 CustomXMLPart()
    Dim WordApp As Word.Application, doc As Word.Document '声明 Word 应用程序、文档
    Dim p As Office.CustomXMLPart                          '声明一个自定义 XML 部分
    Dim i As Integer
    Set WordApp = GetObject(, "Word.Application")          '获取 Word 应用程序
    Set doc = WordApp.Documents("文档中的自定义 XML.docx") '获取 Word 文档
    Set p = doc.CustomXMLParts.Add()                    '为 Word 文档添加一个自定义 XML
```

```
    p.Load (ThisWorkbook.Path & "\ 西南省份 .xml")          ' 装载文件
    Set p = doc.CustomXMLParts.Add()
    p.Load (ThisWorkbook.Path & "\ 东北地区 .xml")
    For i = 4 To doc.CustomXMLParts.Count
        Debug.Print i, doc.CustomXMLParts.Item(i).XML  ' 打印每个自定义 XML 的代码
    Next i
    doc.Save
End Sub
```

运行上述过程后，在 Word 中关闭“文档中的自定义 XML.docx”文件，然后用 WinRAR 打开，如图 4-25 所示。

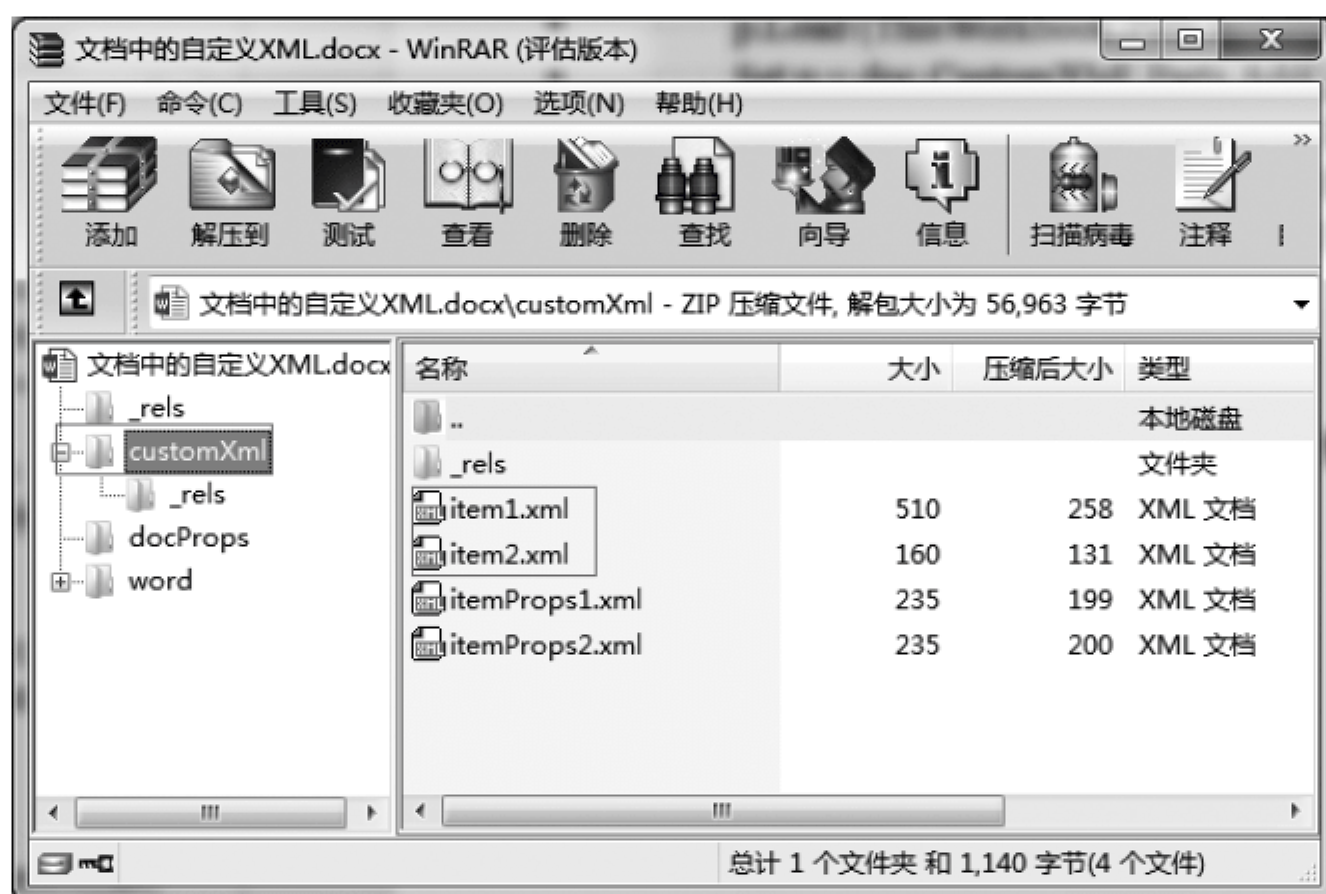


图 4-25　用 WinRAR 打开 Word 文档

在 WinRAR 中可以看到该 Word 文档多了一个 customXML 文件夹，其中有 item1.xml 和 item2.xml 两个文件，这两个就是利用 CustomXMLParts.Add 方法添加进来的。

4.10.2　读取 Office 文档中的自定义 XML

对于存储于 Office 文档中的自定义 XML，还可以用同样的方法读取出来，赋给 DOMDocument 对象。

下面的过程读取 Word 文档中包含的两个自定义 XML。首先手工在 Word 中打开文档“文档中的自定义 XML.docx”，然后执行 Excel VBA 中的如下过程。

```
Sub 读取 CustomXMLPart()
    Dim WordApp As Word.Application, doc As Word.Document ' 声明 Word 应用程序、文档
    Dim p As Office.CustomXMLPart                         ' 声明一个自定义 XML 部分
    Dim DOM As MSXML2.DOMDocument
    Set WordApp = GetObject(, "Word.Application")         ' 获取 Word 应用程序
    Set doc = WordApp.Documents(" 文档中的自定义 XML.docx") ' 获取 Word 文档
    For i = 4 To doc.CustomXMLParts.Count
        Set p = doc.CustomXMLParts.Item(i)
        Set DOM = New DOMDocument
        DOM.LoadXML p.XML
        Debug.Print DOM.XML
    Next i
```

```
    doc.Save
End Sub
```

运行上述过程，立即窗口成功打印出 XML 代码。

4.10.3 移除 Office 文档中的自定义 XML

下面的过程移除 Word 文档中的自定义 XML。

```
Sub 移除 CustomXMLPart()
    Dim WordApp As Word.Application, doc As Word.Document '声明 Word 应用程序、文档
    Dim p As Office.CustomXMLPart                         '声明一个自定义 XML 部分
    Set WordApp = GetObject(, "Word.Application")         '获取 Word 应用程序
    Set doc = WordApp.Documents("文档中的自定义 XML.docx") '获取 Word 文档
    For i = 4 To doc.CustomXMLParts.Count
        Set p = doc.CustomXMLParts.Item(4)
        p.Delete
    Next i
    doc.Save
End Sub
```

代码分析：为了不移除内置的 XML，For 循环从 4 向后遍历，多次移除第 4 个自定义 XML，就可以移除所有自定义 XML。

运行上述代码后，再用 WinRAR 打开“文档中的自定义 XML.docx”，就看不到 customXml 文件夹了，如图 4-26 所示。

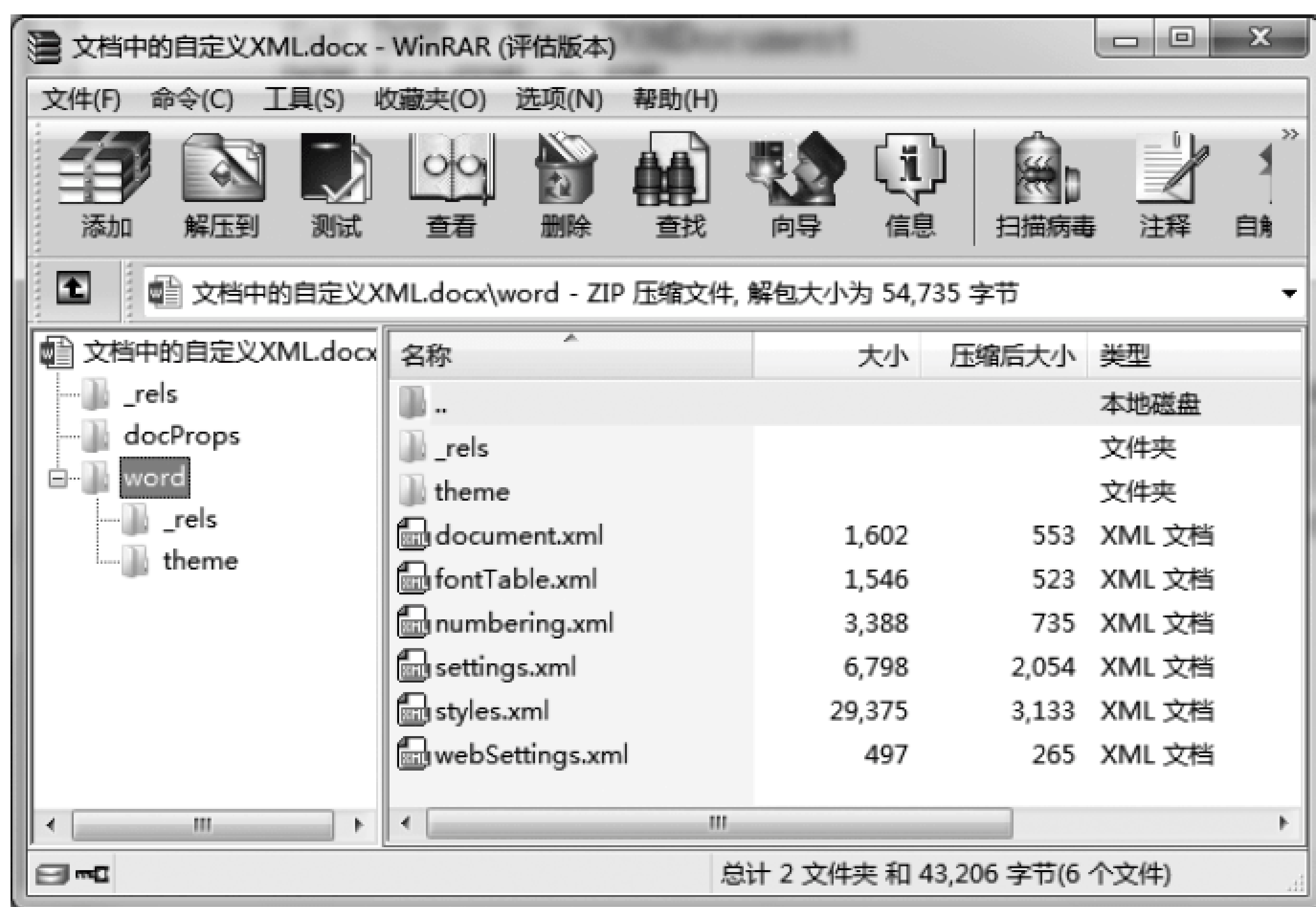


图 4-26 移除 Office 文档中的 XML 部分

需要注意的是，本节讲述的自定义 XML 与后面讲到的自定义功能区不是一回事。

4.10.4 工作表导入 XML

Excel VBA 的 Workbook 对象有 XmlMaps 和 XmlMap 对象，使用 Workbook.XmlImport 方法可以把网络的 XML 文件或本地 XML 文件导入工作簿，成为一个 XmlMap 对象。

Workbook 的 XmlImportXml 方法与 XmlImport 方法非常类似，不同的是前者接受的参数是 XML 代码字符串，后者是一个文件路径。

下面的程序向工作簿中添加一个 XmlMap 对象，XML 的源来自于网络 url，并且把导入的内容显示于单元格 A1。

```
Sub 工作表导入 XML 文件 ()
    Dim Map As Excel.XmlMap
    Dim code As String
     ActiveWorkbook.XmlImport URL:="http://www.w3school.com.cn/example/xmle/simple.xml", ImportMap:=Nothing, Overwrite:=True, Destination:=ActiveWorkbook.Worksheets(1).Range("A1")
    Application.DisplayXMLSourcePane
    Debug.Print ActiveWorkbook.XmlMaps.Count
    Set Map = ActiveWorkbook.XmlMaps.Item(1)
    Map.ExportXml Data:=code
    Map.Export URL:=ActiveWorkbook.Path & "\simple.xml", Overwrite:=True
End Sub
```

代码分析：XmlMap 对象还具有 Export 方法，可以把存在于工作簿中的 XmlMap 导出到字符串变量或者文件中。

运行上述程序，当前工作簿中导入了相应的 XML 数据，并且自动显示 XML 源窗格，如图 4-27 所示。

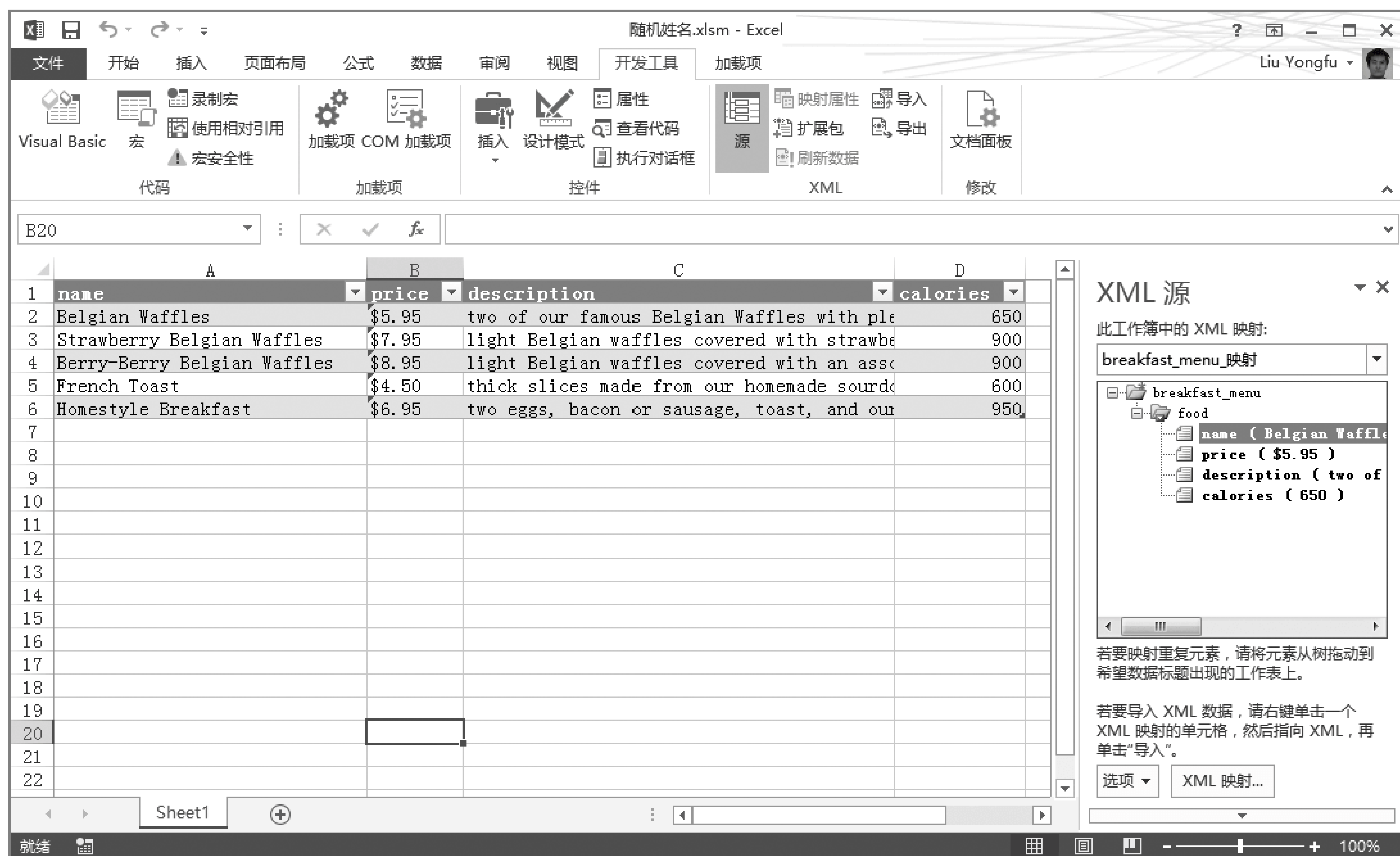


图 4-27 工作表中导入 XML

以上内容的源代码文件为“实例文档 15.xlsm”。

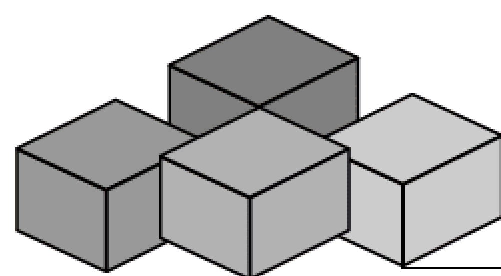
4.11　本章小结

XML 文件中，一切皆为节点，其中元素这种节点构成了 XML 文件的骨架。

XML 的 DOMDocument 文档对象的 Load 方法用于装载一个 XML 文件，而 LoadXML 用于装载一个字符串。Save 方法可以把 DOMDOcument 对象保存为 XML 文件。

DOMDocument 文档对象下面有一系列以 Create 开头的方法，用于创建节点。但是创建了的节点还没有加入文档树中，需要用 AppendChild 方法附加进去。

节点的 AppendChild 方法可以使另一个节点成为子节点。RemoveChild 方法可以移除一个子节点。



第 5 章

自定义功能区

微软 Office 自 2007 版开始使用功能区代替传统的菜单栏、工具栏，而且允许开发人员对 Office 界面进行自定义。

通过学习本章知识，可以在 Office 界面加入自定义的控件，并且能够调用 VBA 中的过程，从而制作出更加专业的插件和工具。

5.1　customUI 概述

用户自定义界面（Custom User Interface，简称 customUI），使用 XML 语言来描述。XML 代码中通过元素节点的包含关系来表达控件的包含关系，用元素节点的属性来表达真正控件的属性。

customUI 的 XML 代码可以存储于 Excel 工作簿、Word 文档、PowerPoint 演示文稿或 Access 数据库中。但是对于没有文档的组件，例如 Outlook，只能在 COM 外接程序中使用 customUI 的 XML。本书只讨论压缩于 Office 文档中的 customUI 的技术。

customUI 开发过程包含如下三个层面。

- ❑ XML 代码的编写和存储。
- ❑ Office 界面呈现。
- ❑ VBA 回调的生成和执行。

其实，可以用一句话来概括 customUI 的开发过程：

“修改 XML 代码，使得 Office 能够呈现出期望的界面，并且单击界面中的自定义控件，能够调用 VBA 中的宏。”

可以看出，customUI 的开发过程，实际上就是书写和优化 XML 代码与 VBA 代码的过程。

Office 界面中，可以自定义的部分，也就是允许把自定义控件加入的场所，主要有以下 5 个地方。

- ❑ 常用功能区（tabs）。
- ❑ 快速访问工具栏（qat）。

- ❑ 环境功能区（contextualTabs）。
- ❑ 右键菜单（contextMenus）。
- ❑ Office 菜单（backstage）。

5.1.1　常用功能区

常用功能区是指 Office 上方的长条区域，由多个选项卡组成，每个选项卡包含多个组，每个组可以有多个控件，如图 5-1 所示。

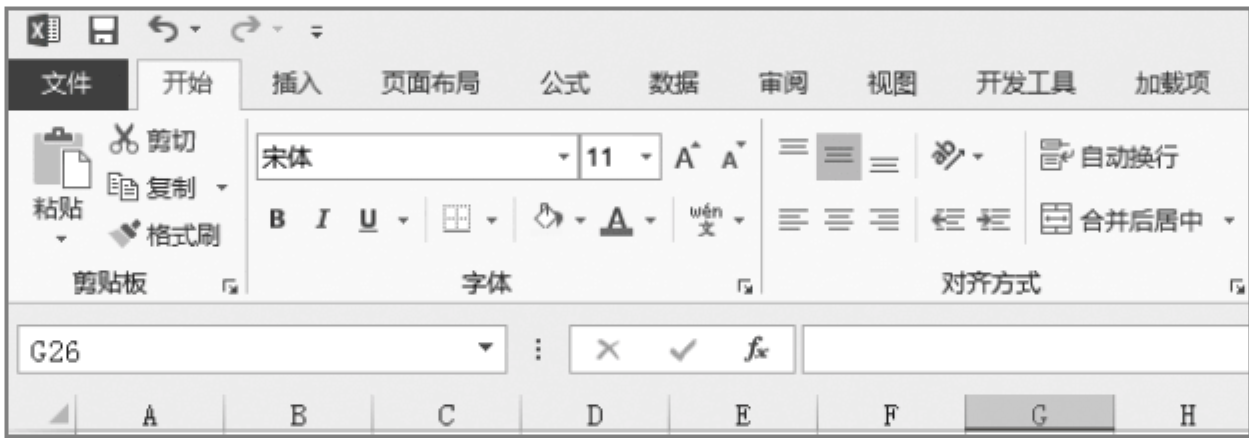


图 5-1　Excel 常用功能区

例如，在 Excel 2013 的常用功能区中，“开始”“插入”等选项卡构成了一个 tabs 集合，这些选项卡是切换显示的，也就是说不可能同时看到多个选项卡。

“开始”选项卡下面包含“剪贴板”“字体”等组（group），每个组中包含多种控件。

5.1.2　快速访问工具栏

快速访问工具栏（qat）是指位于常用功能区上方的很窄的区域，默认包括“保存”“撤销”“重做”等基本命令。Office 允许用户在快速访问工具栏中添加和删除命令。

5.1.3　环境功能区

环境功能区（contextualTabs），是指鼠标选中某类型对象，例如选中了一个图片，常用功能区的右侧会出现临时的选项卡。当鼠标未选中任何图片时，该选项卡消失，如图 5-2 所示。

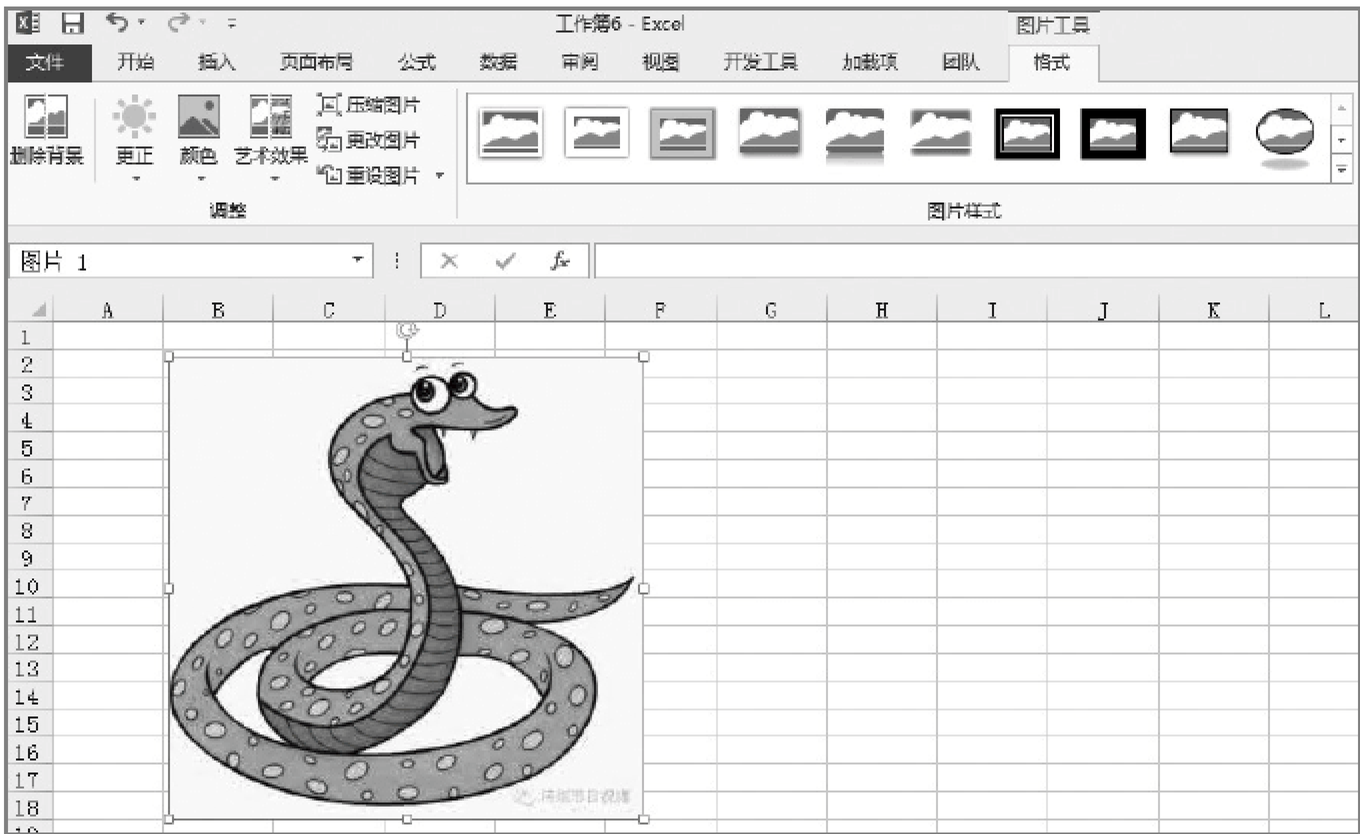


图 5-2　特定对象的环境功能区

这些临时的选项卡与当前所选对象类型有关。

5.1.4　右键菜单

鼠标选中某个对象并右击，会弹出相应的菜单（contextMenus）。例如 Excel 的工作表标签的右键菜单如图 5-3 所示。

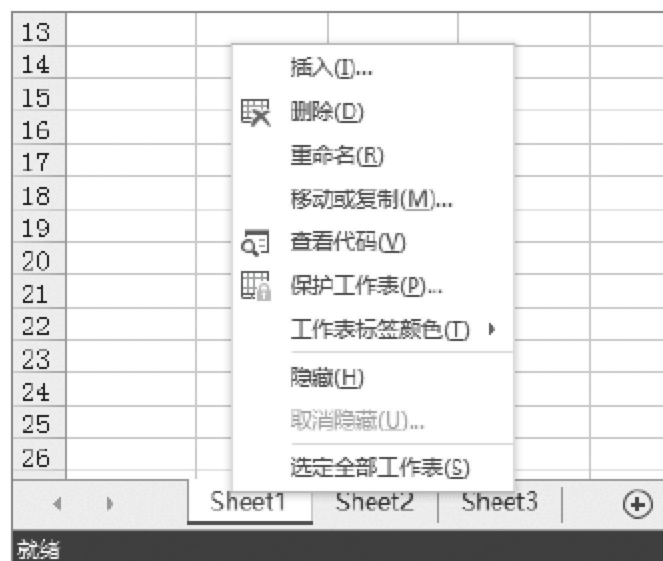


图 5-3　工作表标签右键菜单

5.1.5　Office 菜单

Office 菜单是指鼠标单击 Office 左上角的“文件”后弹出的界面，如图 5-4 所示。



图 5-4　Office 菜单

5.1.6　手动完成 customUI 设计

严格地讲，customUI 开发设计工作，只需安装 Office 以及压缩软件就够了。下面讲解向 Office 文档中压入 customUI 的 XML 代码的方法。

通过第 4 章的内容可以了解到，Office 2007 以上版本的 Excel 工作簿、PowerPoint 演示文稿、Word 文档的扩展名通常是 4 个字母，这些文档实际上都是压缩包，都可以使用 WinRAR 等软件打开、查看和编辑。

首先新建一个空白工作簿，保存为“无 customUI.xlsm”，这是一个非常普通的工作簿，在 Excel 中打开以及关闭该工作簿，不会引起任何的界面变化。

工作簿在关闭的前提下，用 WinRAR 打开该工作簿，可以看到里面包含 3 个文件夹和 1 个 XML 文件，如图 5-5 所示。

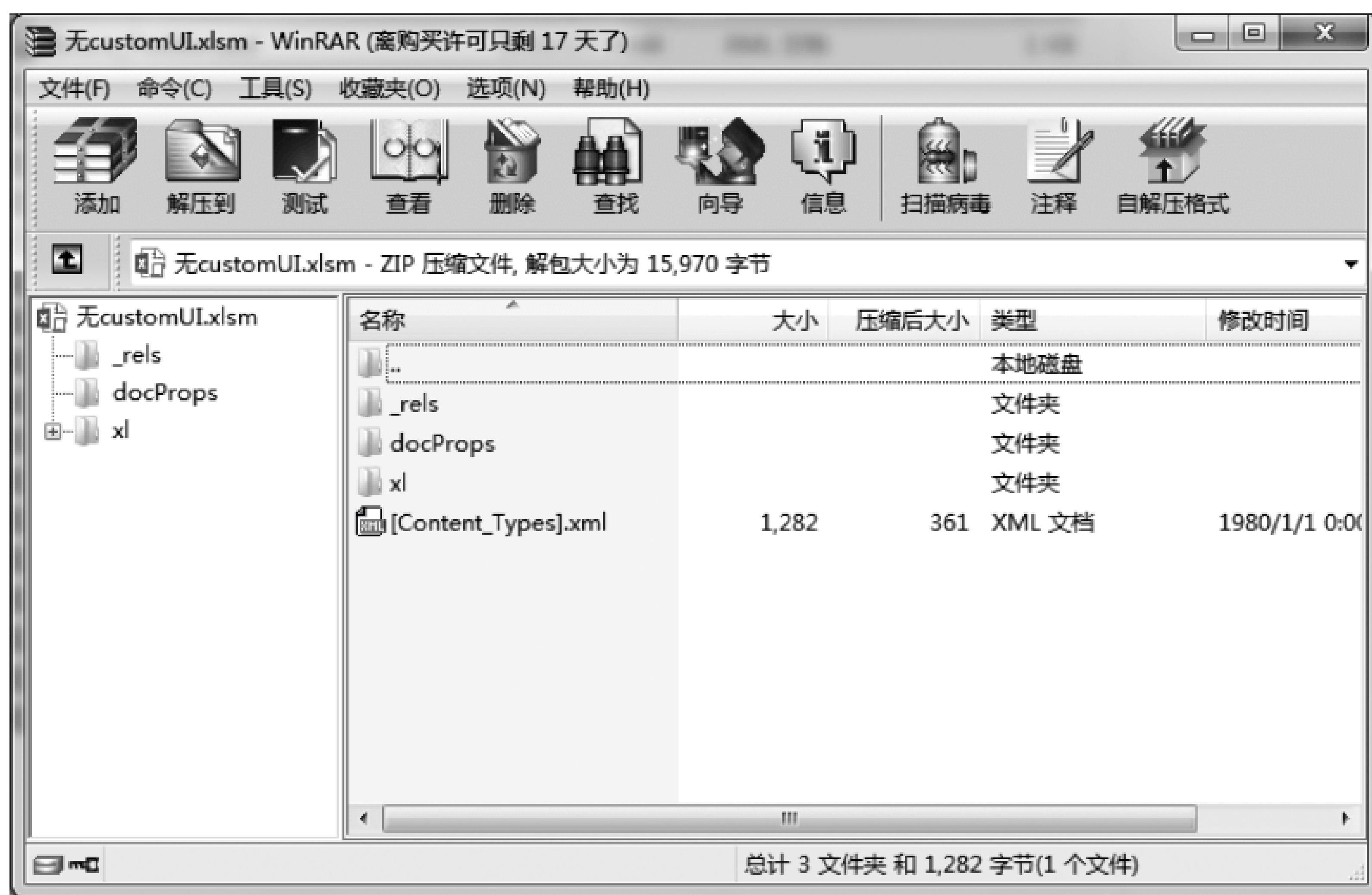


图 5-5　用 WinRAR 打开 Excel 文件

其中，名称为 _rels 的子文件夹中包含着一个名称为 .rels 的 XML 文件，该文件的内容如图 5-6 所示。

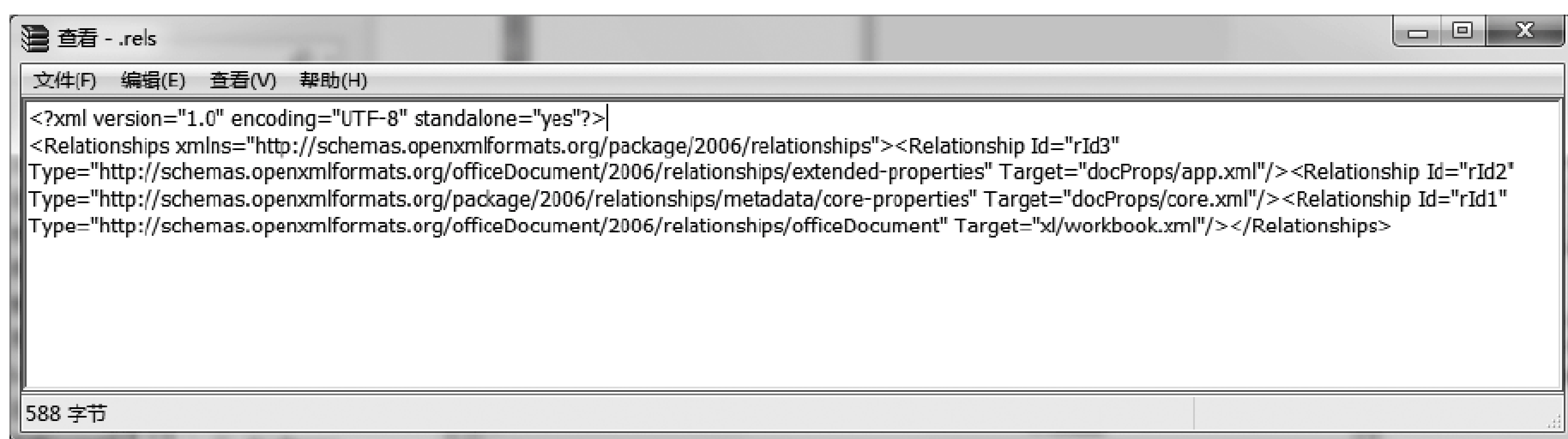


图 5-6　.rels 文件的内容

以上就是不含任何 customUI 的文档的内部内容。

接下来，在计算机的任意一个路径下新建一个名为 customUI 的文件夹，在该文件夹中新建一个文本文档，文档内容如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label="Microsoft" insertAfterMso="TabHome">
                <group id="Group1" label="Office">
                    <button id="ButtonID3" label="VBA" imageMso="V" size="large"
onAction="Button_Click"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

把该文本文档重命名为 customUI14.xml。

最后，把包含 customUI14.xml 的文件夹 customUI 拖曳到 WinRAR 压缩软件窗口中，也就是把该文件夹添加到“无 customUI.xlsm”中，如图 5-7 所示。



图 5-7 把 customUI 文件夹整体压入 Excel 文件

接下来，还需要改动压缩包中的 .rels 文件，首先把该文件释放到计算机的任意路径，用记事本程序修改为如图 5-8所示的样子。

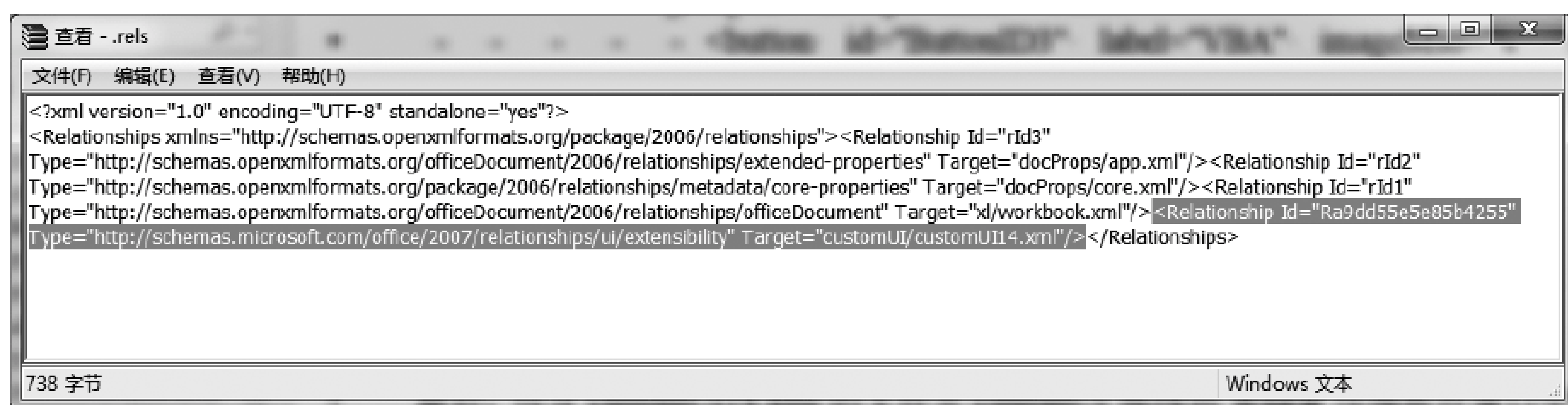


图 5-8 修改 .rels 文件内容

其中，反选的那行文字：

```
<Relationship Id="Ra9dd55e5e85b4255" Type="http://schemas.microsoft.com/office/2007/relationships/ui/extensibility" Target="customUI/customUI14.xml"/>
```

就是告诉 Excel，customUI 使用的文件是 customUI14.xml。

修改后的文件再次拖放回压缩包进行同名文件替换，此时就可以关闭压缩工具，在 Excel 中打开该工作簿，可以看到在“开始”选项卡右边多了一个新选项卡，单击“VBA”按钮，弹出无法运行宏的对话框，如图 5-9 所示。

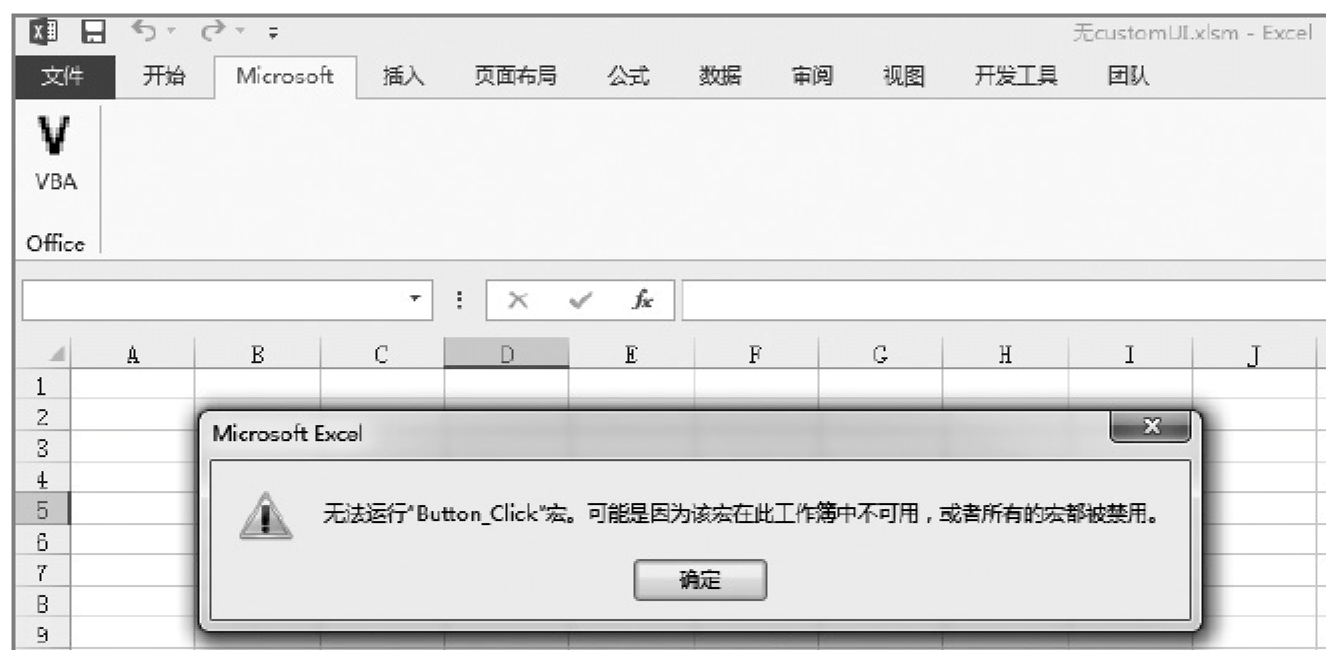


图 5-9 customUI 找不到对应的回调函数

这是因为前面的 XML 代码中有一句：

```
<button id="ButtonID3" label="VBA" imageMso="V" size="large" onAction="Button_Click"/>
```

这就要求在 VBA 中有一个名为 Button_Click 的宏，因此在该工作簿的 VBA 工程中插入一个标准模块，模块中的内容如下。

```
Public Sub Button_Click(control As Office.IRibbonControl)
    MsgBox ActiveCell.Address
End Sub
```

这个 VBA 过程就称作按钮的回调（callback）。

回到 Excel 再次单击“VBA”按钮，会弹出一个显示单元格地址的对话框。

以上手工定制的方法也适用于 PowerPoint 演示文稿和 Word 文档。

如果对界面不满意，还可以继续使用 WinRAR 打开文档，编辑并替换 customUI14.xml 这个文件，不断修改完善。

注意　存储于文档中的 customUI 部分，只有该文档打开，并且处于活动文档时，才能看到自定义界面，如果该文档失去焦点或者被关闭，自定义界面随之消失。

5.2　使用 customUI 软件

通过手工进行 customUI 的设计，显然比较烦琐，通常借助专业软件来辅助完成。常用的 customUI 设计软件和工具如表 5-1 所示。

表 5-1　常用的 customUI 软件和工具

软 件 名 称	支持 Schema 验证	自动成员提示	自动生成回调函数	压 入 文 档
Custom UI Editor	●		●	●
Office Ribbon Editor	●	●		●
VS XML Editor	●	●		
Ribbon XML Editor	●		●	●

5.2.1　命名空间和 Schema 验证

对于 customUI 中的 XML 编写，微软提供了 Office 2007 和 Office 2010 以上版本的命名空间和 Schema 验证文件。

- Office 2007 兼容。

命名空间：<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">

验证文件：customUI.xsd

- Office 2010 以上版本兼容。

命名空间：<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">

验证文件：customUI14.xsd

这些文件的作用就是限制 XML 文件中哪些元素可以出现，哪些属性可以出现，以及属性取值有哪些，如果使用了不符合验证文件的内容，验证就不会成功。

因此，无论你用的是哪一款 customUI 软件，一般都把上述文件作为验证标准。

5.2.2 Custom UI Editor

该软件可以向 Office 文档中压入 XML，支持自定义图标，总体性能良好。但是在书写 XML 代码时，没有自动成员提示，如图 5-10 所示。

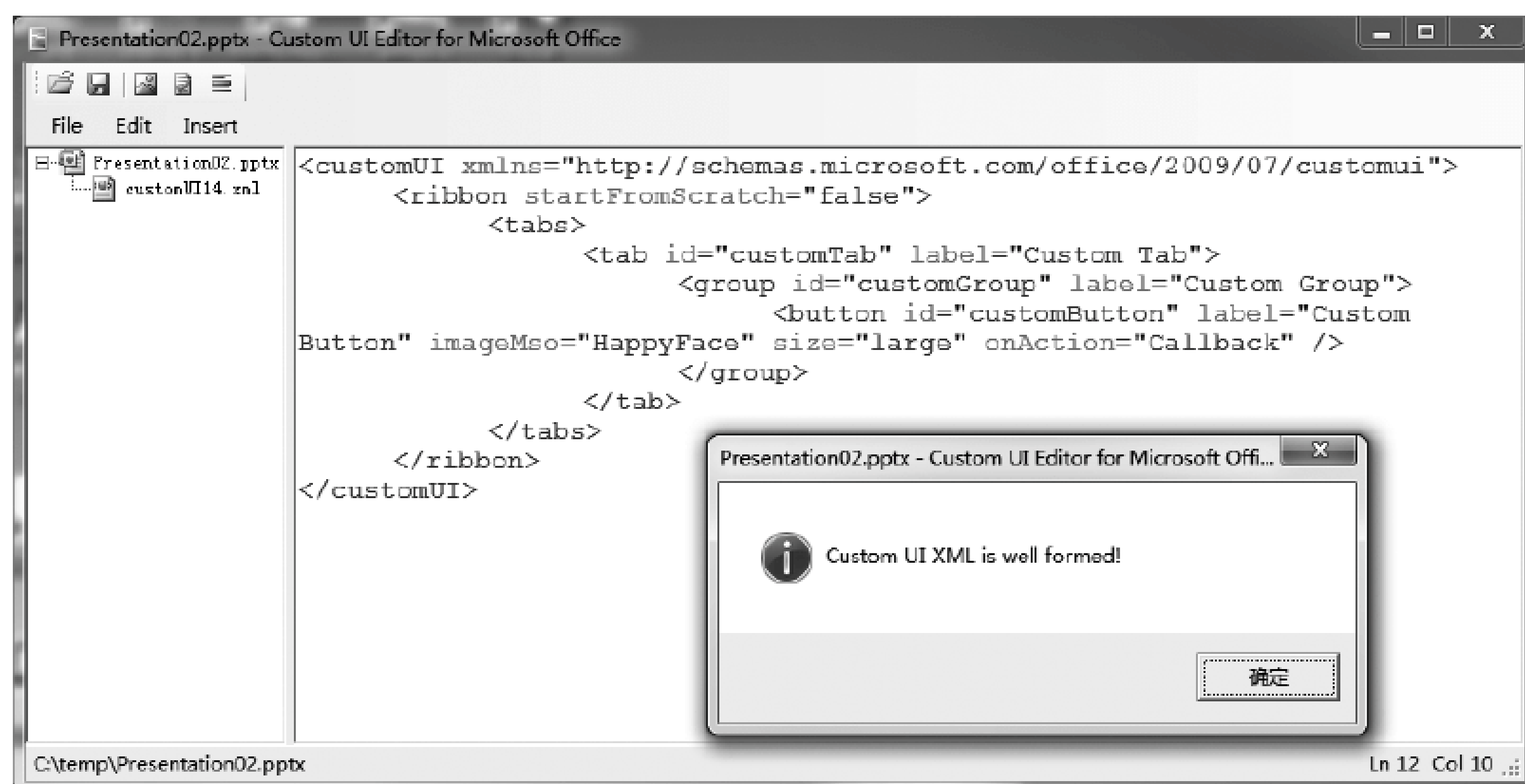


图 5-10 Custom UI Editor 软件弹出错误提示

5.2.3 Office Ribbon Editor

该软件的功能更加丰富，有自动成员提示，如图 5-11 所示。

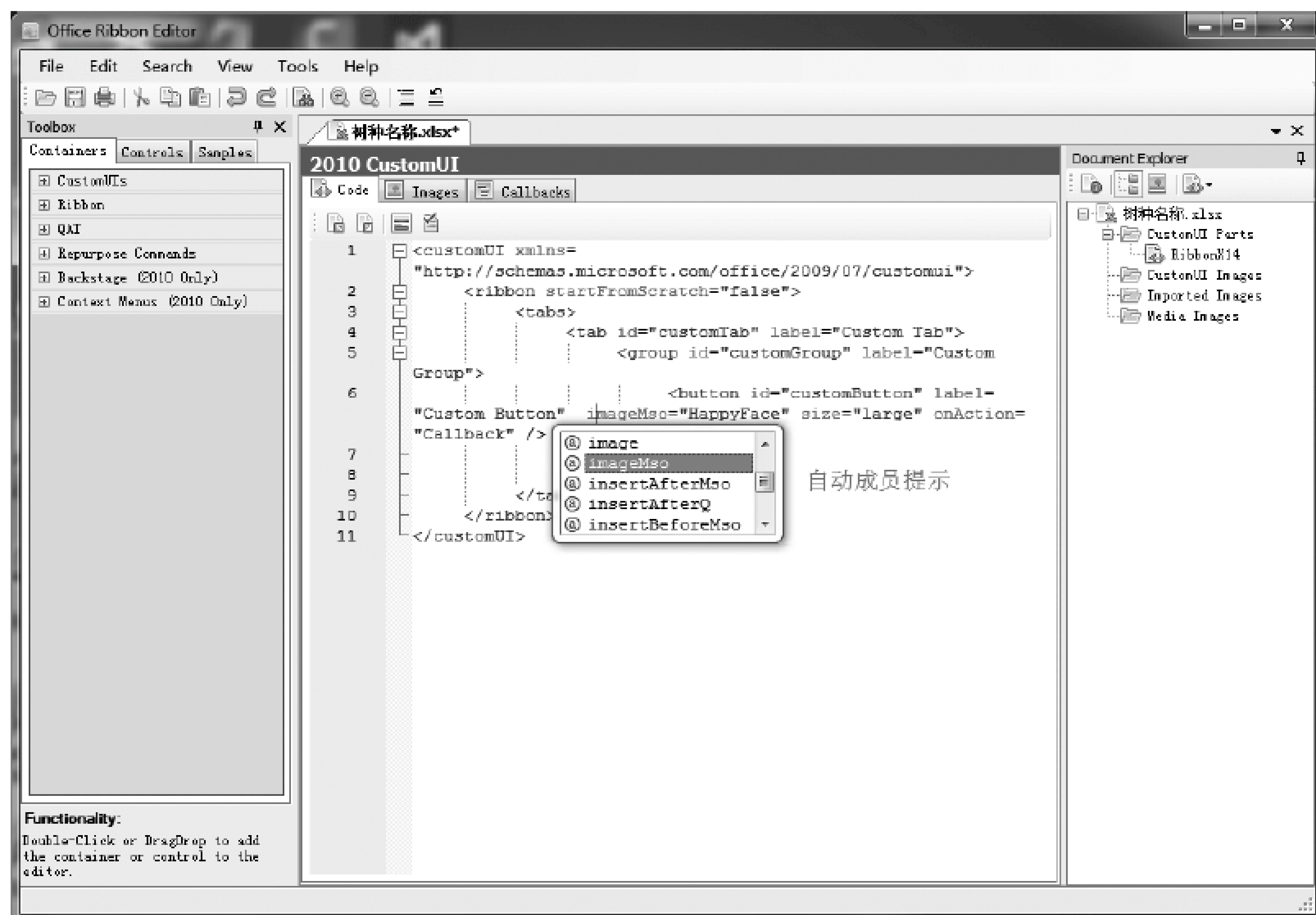


图 5-11 Office Ribbon Editor 软件有自动成员提示

5.2.4 Visual Studio 中的 XML Editor

在 Visual Studio 中，单击菜单【文件 / 打开文件】，浏览到任意一个 XML 文件，就可以

进行编辑。

只要该 XML 文件的命名空间是 Office customUI 的命名空间，按下 < 键就会弹出该位置允许的元素名称列表，如图 5-12 所示。

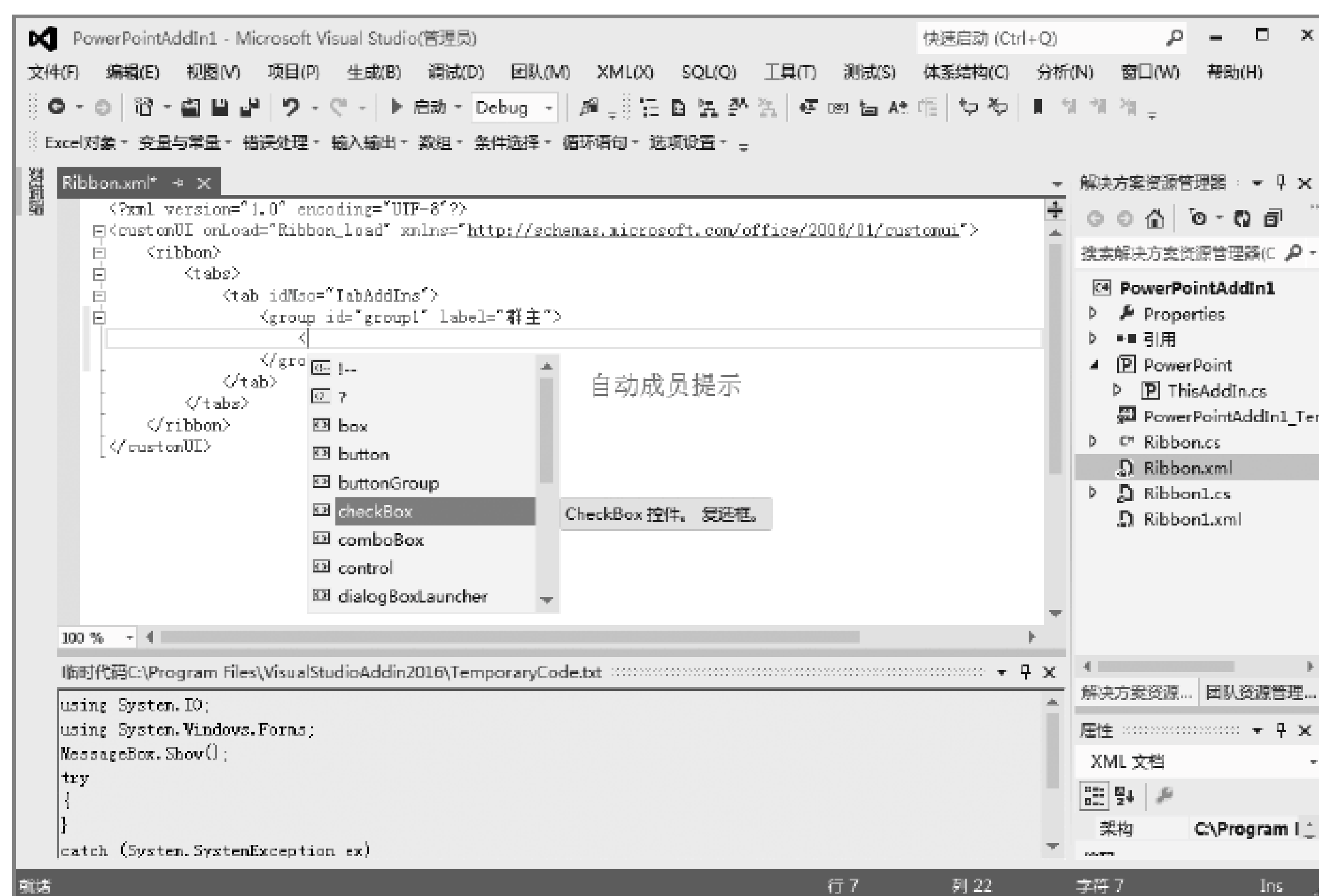


图 5-12　Visual Studio 的 XML 编辑器

在书写 XML 时，可以明确地知道元素下面允许使用哪些子元素，元素可以包含哪些属性，属性可以取哪些值。

5.2.5　Ribbon XML Editor

该软件是笔者开发的一款功能强大的 customUI 工具，如图 5-13 所示。

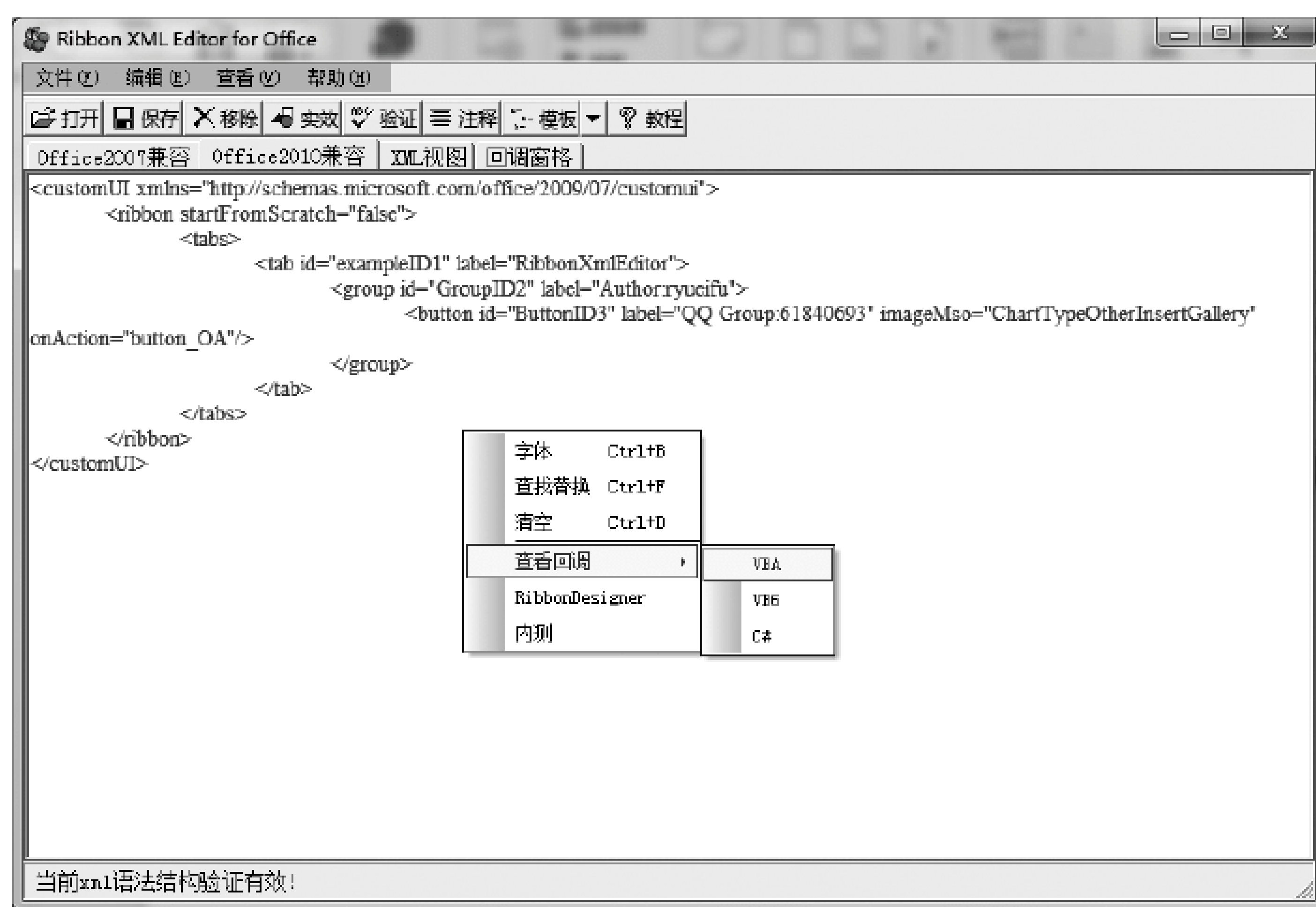


图 5-13　Ribbon XML Editor 软件界面

除了具有其他软件的通用功能外，本工具还具有以下几个独特功能。

- 实时查看功能：无须保存到 Office 文档，就可以立即看到 Office 界面的变化。
- 面向 VBA、VB6、VSTO 开发的多种回调函数。
- 自动补全、自动缩进功能。

以从本书配套资源下载 RibbonXMLEditor2018.01.06-Setup.exe。

本章以下内容均采用 Ribbon XML Editor 软件进行 customUI 设计。因此首先介绍一下该软件的基本用法。

1. 向 Office 文档压入 XML

启动 Ribbon XML Editor 软件，切换到“Office2010 兼容”，输入如图 5-14 所示的 XML 代码。

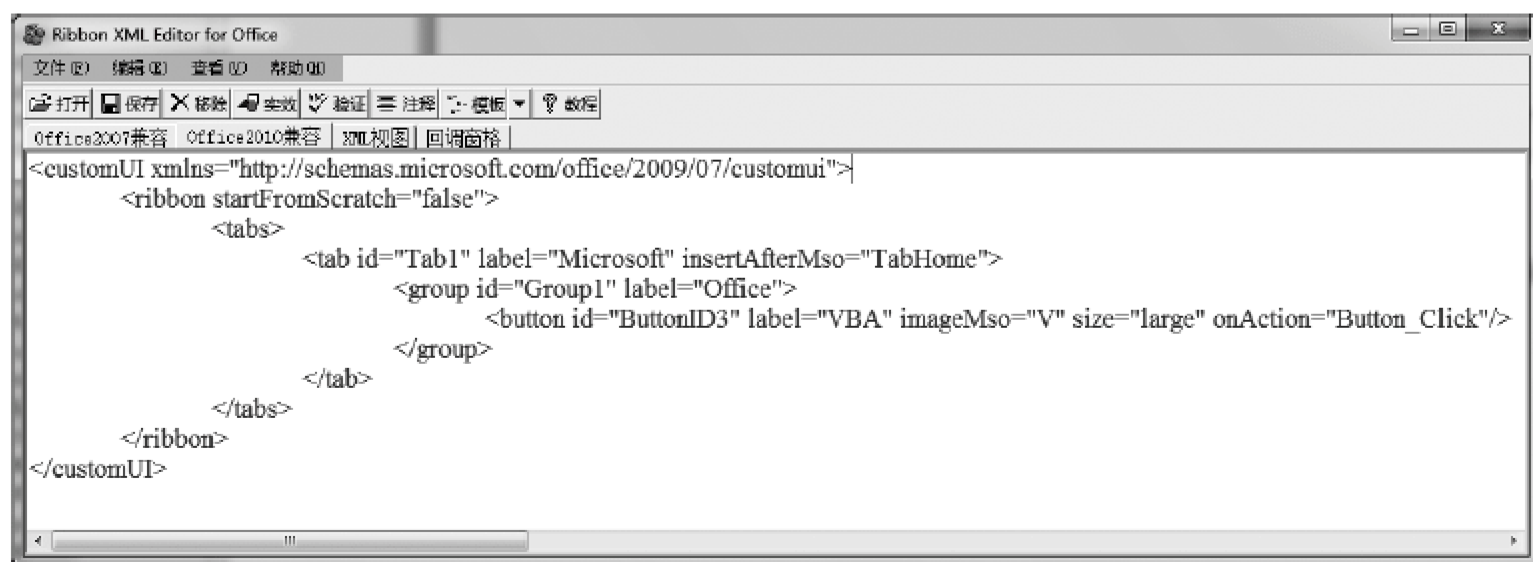


图 5-14　编辑 XML 代码

单击工具栏中的“验证”按钮，如果验证成功，单击“保存”按钮，选择计算机中已有的一个 Office 文档，例如“实例文档 31.xlsm”。提示压入成功，在 Excel 中再次打开该文档，在“公式”选项卡右侧出现一个新的自定义选项卡，如图 5-15 所示。

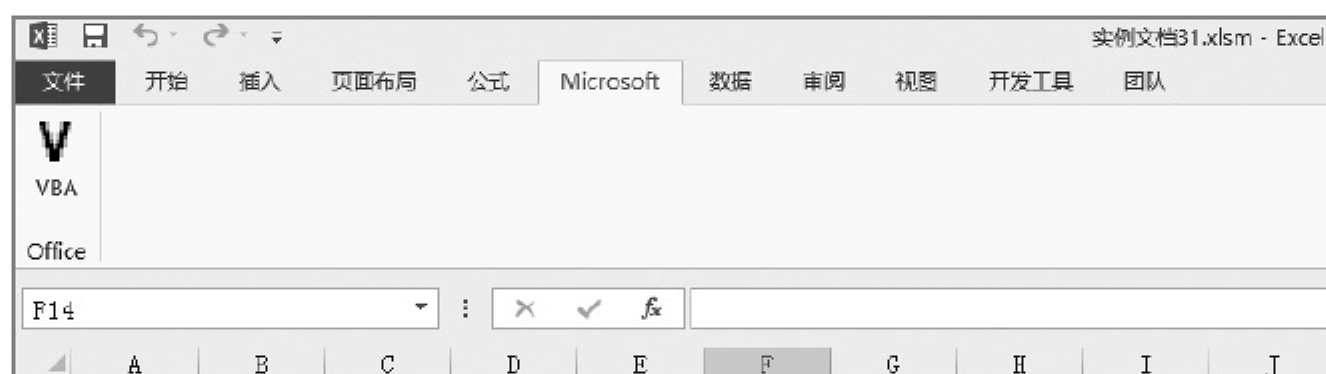


图 5-15　打开带有 customUI 的 Excel 文件

接下来，为 button 设计回调函数，回到 Ribbon XML Editor，在代码区右击，在弹出菜单中选择【查看回调 /VBA】，此时自动得到回调函数，如图 5-16 所示。

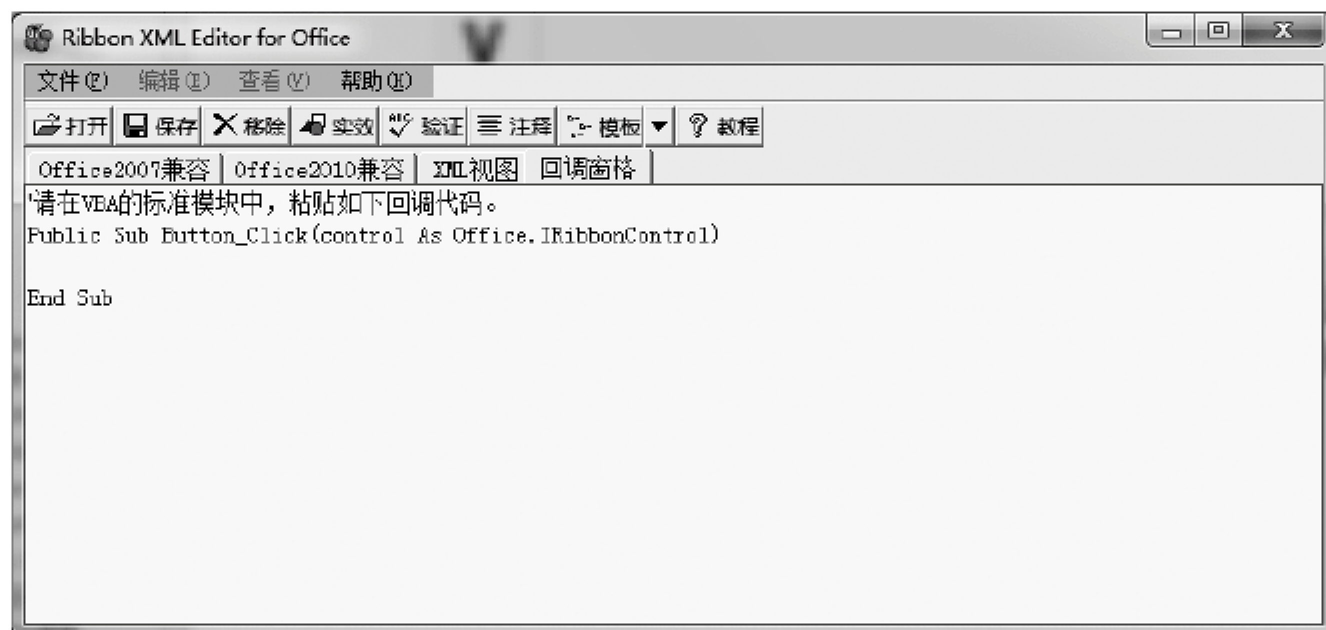


图 5-16　自动得到回调函数

把上述过程粘贴到工作簿的标准模块中，并增加过程中的代码，如下所示。

```
Public Sub Button_Click(control As Office.IRibbonControl)
    MsgBox ActiveCell.Address
End Sub
```

此时，单击功能区中的自定义按钮，会弹出活动单元格地址。

可以看出，一个完整的 customUI 设计的流程包括以下 5 个必不可少的步骤。

（1）编写 XML 代码。

（2）验证 XML 代码。

（3）查看回调函数。

（4）把 XML 压入 Office 文档。

（5）打开 Office 文档查看界面，并写入回调函数，保存文件。

2. 查看现有文档中的 customUI 部分

Ribbon XML Editor 也可以从现有文档中抽出 XML 代码，单击工具栏的“打开”按钮，选中一个 Office 文档，就可以取出该文档中包含的 XML 代码。

如果单击工具栏中的“移除”按钮，则可以把包含 customUI 的 Office 文档中的 XML 全部移除。

3. 实时查看效果

Ribbon XML Editor 软件可以在编写 XML 的同时，直接在相应的 Office 组件中查看效果。例如，单击菜单【查看 /PowerPoint】，或者按下快捷键【Ctrl+F4】，如图 5-17 所示。

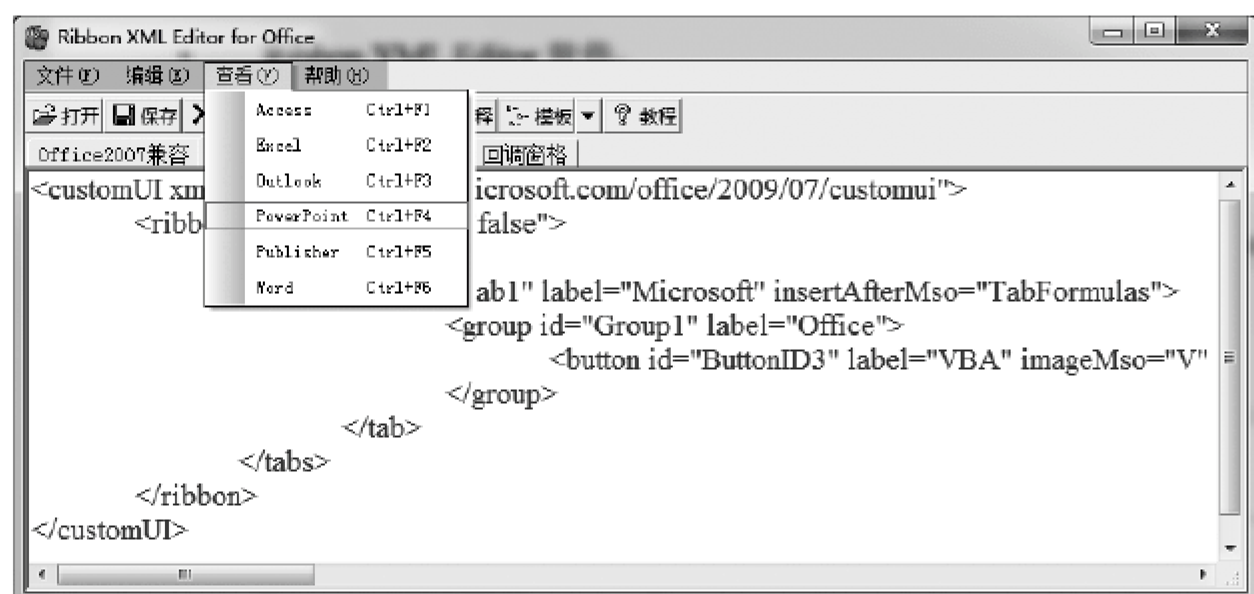


图 5-17　在相应组件中查看 customUI 效果

此时，会自动启动 PowerPoint，并可以看到在 PowerPoint 中多了自定义的界面，如图 5-18所示。



图 5-18　PowerPoint 中查看 customUI 效果

这样，无须向 Office 文档中保存 XML，就可以快速看到 customUI 引起的界面变化。

5.2.6 显示加载项用户界面错误

编写 XML 时，经常遇见各种各样的错误，例如 idMso 控件名称不存在、未在 VBA 中建立相应的回调函数等，如果把不合适的 XML 压入 Office 文档，打开该文档应该会弹出相应的错误。

因此，需要在 Excel 的选项对话框中切换到“高级”选项卡，勾选“显示加载项用户界面错误”，如图 5-19 所示。

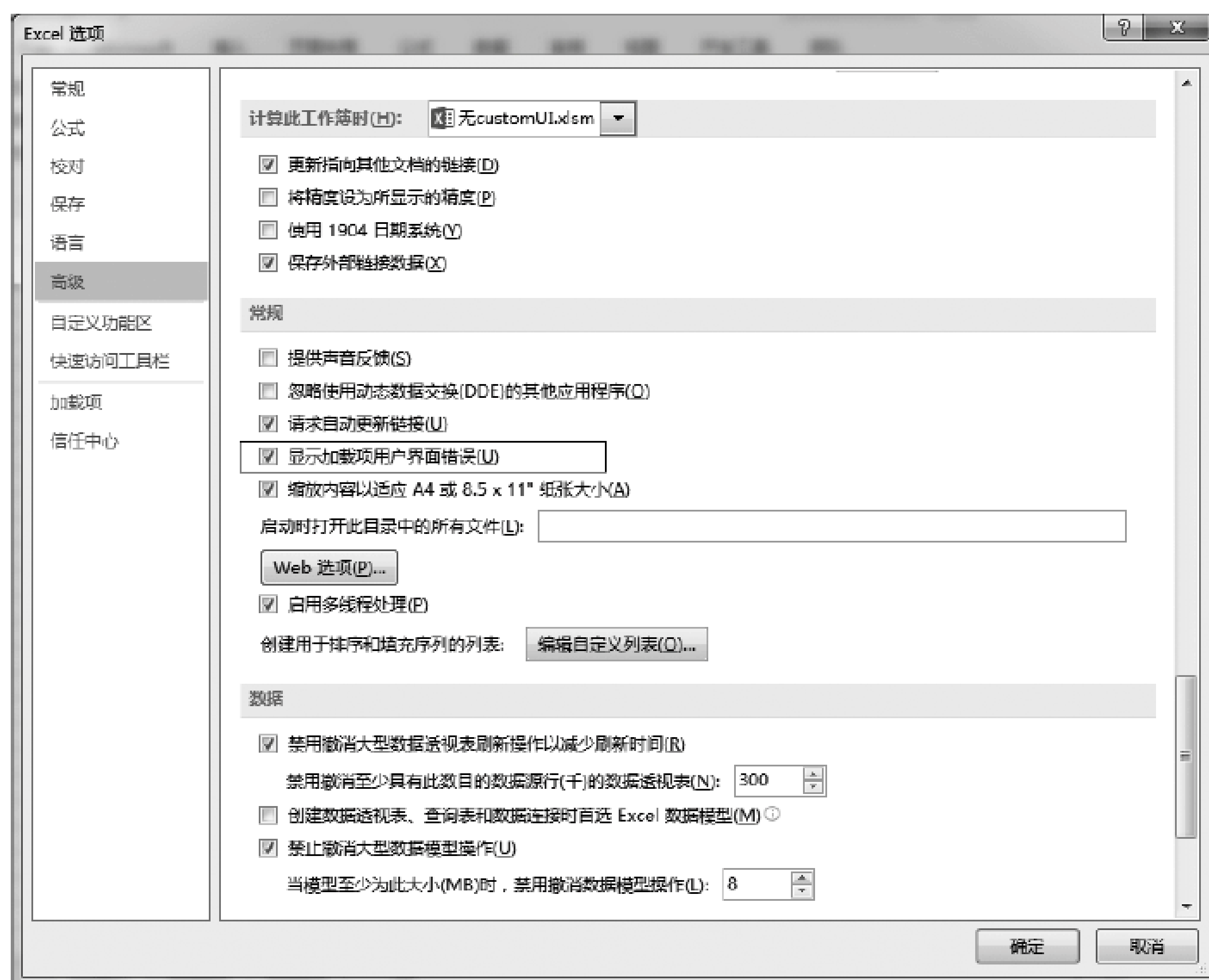


图 5-19 设置“显示加载项用户界面错误”

如果不勾选，即使 XML 存在严重错误，文档打开后，也不会有任何提示。因此，为了顺利进行 customUI 开发，一定要勾选该复选框。

例如，在 Excel 中应用如下 customUI。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label="Microsoft" insertAfterMso="TabFormula">
                <group id="Group1" label="Office">
                    <button id="ButtonID3" label="VBA" imageMso="V" size="large"
onAction="Button_Click"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

此时会立即弹出“未知 Office 控件 ID：TabFormula”错误，如图 5-20 所示。

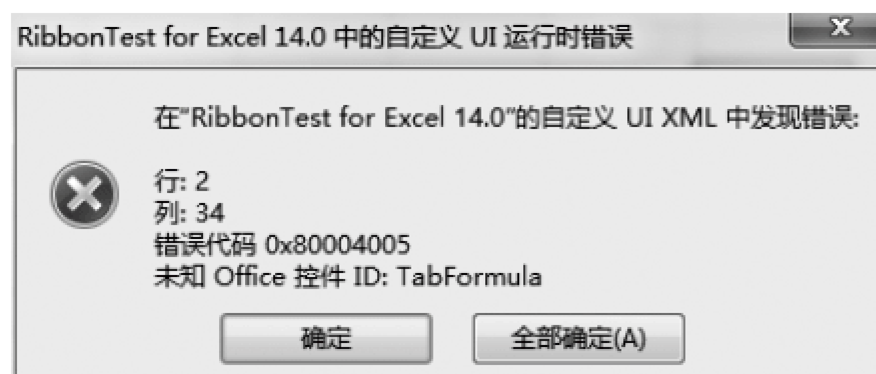


图 5-20 customUI 引起的错误

引起上述错误的原因是，Excel 的“公式”选项卡的 idMso 是 TabFormulas。

5.3　自定义常用功能区

常用功能区是指 Office 软件上面的条形区域，通常位于快速访问工具栏下方，如图 5-21 所示。



图 5-21　常用功能区

对于 Excel 2013，在常用功能区中由“开始”“插入”等选项卡构成了一个 tabs 集合。对于某一特定选项卡，例如，“开始”选项卡由“剪贴板”“字体”“对齐方式”等组（group）构成。组与组之间通过一条竖线分隔。

组是由各种各样的控件（control）构成的。

因此，用于自定义常用功能区的 XML 层级结构如图 5-22 所示。

```
- <customUI>
  - <ribbon>
    - <tabs>
      - <tab>
        - <group>
            <control />
          </group>
        </tab>
      </tabs>
    </ribbon>
  </customUI>
```

图 5-22　自定义常用功能区的 XML 结构

用于自定义常用功能区的 XML 只能有一个 customUI、ribbon、tabs 元素节点。但是 tabs 下面可以有多个 tab（选项卡），一个 tab 下面可以有多个 group（组），一个 group 下面可以有多个控件元素。

5.3.1　选项卡

对选项卡（tab）的定制，既可以在内置选项卡后面创建全新的用户选项卡，也可以对 Office 内置选项卡进行修改。

启动 Ribbon XML Editor，切换到选项卡【Office 2010 兼容】，单击工具栏【模板 / 功能区】，自动生成常用功能区的模板，如图 5-23 所示。



图 5-23　使用模板

然后在模板代码中间空白处插入相关代码，效果如下：

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
            </tab>
            <tab idMso="TabHome" label="Start">
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

单击【验证】按钮，通过验证后，单击菜单【查看 /Excel】，会看到 Excel 多了一个“扩展功能”的新选项卡，而且内置选项卡“开始”被重命名为“Start”，如图 5-24 所示。



图 5-24　修改内置选项卡的标题

XML 代码分析：

<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"> 这句声明了命名空间，一般 Office 2010 及其以上版本都是这个命名空间。Office 2007 要换成 <customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">。

startFromScratch="false" 表示不擦除内置选项卡，如果改为 startFromScratch="true"，则表示隐藏内置选项卡，Excel 会变成如图 5-25 所示的样子。

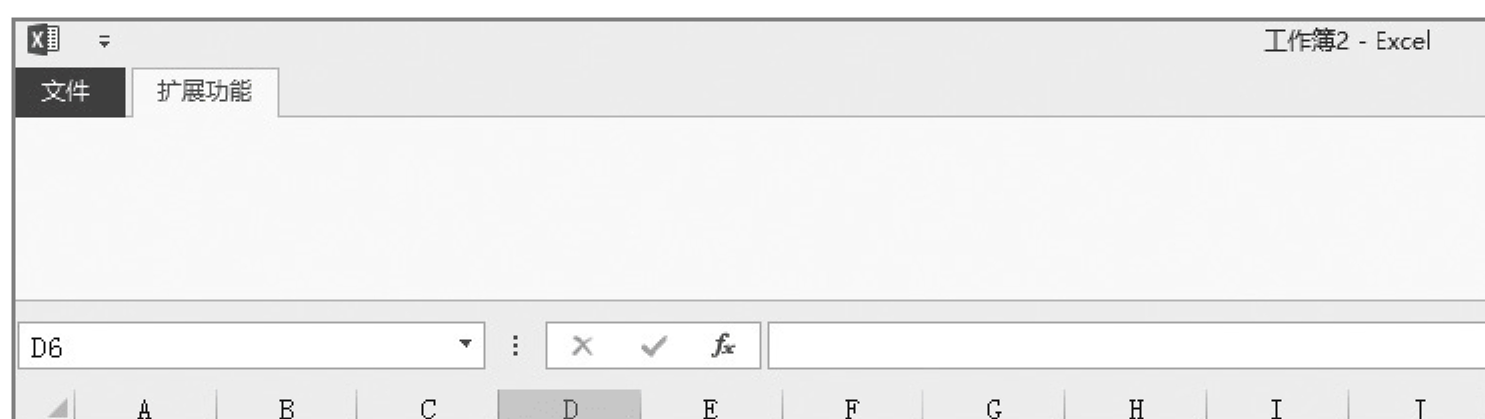


图 5-25　隐藏内置选项卡

<tab id="Tab2018" label=" 扩展功能 "> 表示创建一个新选项卡，指定了 id 和 label。

对于用户创建的自定义 tab、group 和各种控件，必须指定 id，而且互不重复，也就是不能有两个元素的 id 一样。

对于引用 Office 内置 tab、group 和控件，不能使用 id，而是使用 idMso。例如：

```
<tab idMso="TabHome" label="Start">
```

表示把内置选项卡“开始”的标题修改为“Start”。

具体 Office 2013 有哪些内置 idMso，请参考 5.4.1 节。

5.3.2　组

组（group）是 tab 的子元素节点，同时 group 也是各种控件的父节点。用户自定义组既可以放在自定义选项卡中，也可以放在 Office 内置选项卡中。

group 的主要属性与 tab 相似，主要包括 id 和 label 这些常规属性。

下面的 XML 代码在“扩展功能”选项卡中添加一个“信息反馈”组，然后放入一个 Excel 2013 内置的“字体”组。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 信息反馈 ">
                </group>
                <group idMso="GroupFont">
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

在 Ribbon XML Editor 中，单击【查看 /Excel】，可以看到在“开始”选项卡的“字体”组左侧多了一个自定义组，如图 5-26 所示。



图 5-26　插入自定义组

需要注意的是，XML 中先写的元素先出现，后写的元素后出现。由于代码中自定义组在前，因此自定义组排在左侧。

下面将要讲述 group 中可以放置的各类控件（controls）。控件还可以划分为基本控件（simple control）和容器控件（content control），像按钮、文本框控件等这类不可以再包含其他控件的，就属于基本控件或简单控件；而像菜单、分裂菜单、按钮组等通常是其他控件的容器，因此属于容器控件或复杂控件。

5.3.3　按钮

按钮（button）是命令按钮控件，可以放在自定义的 group 或者内置组中。button 的基本属性也是 id（或 idMso）、label。

下面的 XML 中，在 group 元素之下定义了 2 个 button，规定了每个 button 的 id 和标题文字。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label="扩展功能">
                <group id="Group2019" label="信息反馈">
                    <button id="Button1" label="自动转换"/>
                    <button id="Button2" label="手工转换"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上面 XML 代码对应的 Excel界面如图 5-27 所示。

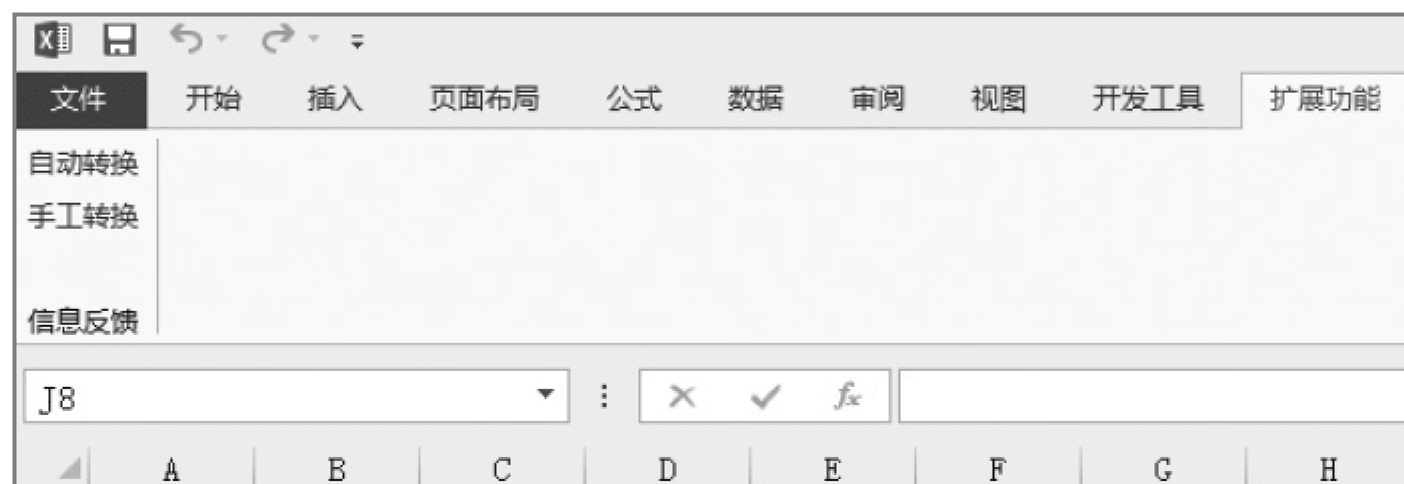


图 5-27　添加自定义按钮控件

可以看出，新的选项卡排在已有选项卡的右侧，组也是这样的。但是像 button 这样的控件，每列能放置 3 个控件，因此按照从上到下先排列，排满一列后向右排列。

需要注意的是，图中“扩展功能”是选项卡的标题文字，“信息反馈”是组的标题文字，“自动转换”是按钮的标题文字。

5.3.4　小结回顾

以上讲述了自定义常用功能区的基本 XML 架构和语法。为了便于后续内容的理解消化，下面列出自定义功能区技术的几个要点和特征。

1. 严格的层次递进关系

常用功能区的 XML 结构如图 5-28 所示。

一定要记住这个递进层次：<customUI><ribbon><tabs><tab><group><controls/>。

其中，controls 表示各种控件。

```
- <customUI>
  - <ribbon>
    - <tabs>
      - <tab>
        - <group>
            <controls />
          </group>
        </tab>
      </tabs>
    </ribbon>
  </customUI>
```

图 5-28　常用功能区的 XML 结构

2. 自定义元素与 Office 内置元素混杂

对于用户新增的元素，例如新的 tab、group、button，必须规定唯一识别的 id 属性，如果被操作或引用的是内置元素，使用 idMso 属性。

3. 容器元素必须用两个标签，简单元素用一个标签

例如 group 是一个容器，那么开始标签是 <group 若干属性 >，然后接着放入其他控件，最后用 </group> 来闭合。

而简单元素则用单行形式形成一个完整的 XML 节点。例如：

```
<button id="Button2" label=" 手工转换 "/>
```

要留意最后面尖括号左侧的斜杠。

下面继续讲解各类型控件的添加方法。

5.3.5　复选框

复选框（checkBox）通常由勾选方块和标题文字构成，可以对用户的勾选和取消勾选行为做出反馈。下面的 XML 在 group 中放入一个自定义 checkBox 控件和一个内置复选框控件。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 复选框展示组 ">
                    <checkBox id="check1" label=" 自动转换 "/>
                    <checkBox idMso="ViewHeadings"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上面 XML 代码对应的 Excel 界面如图 5-29 所示。

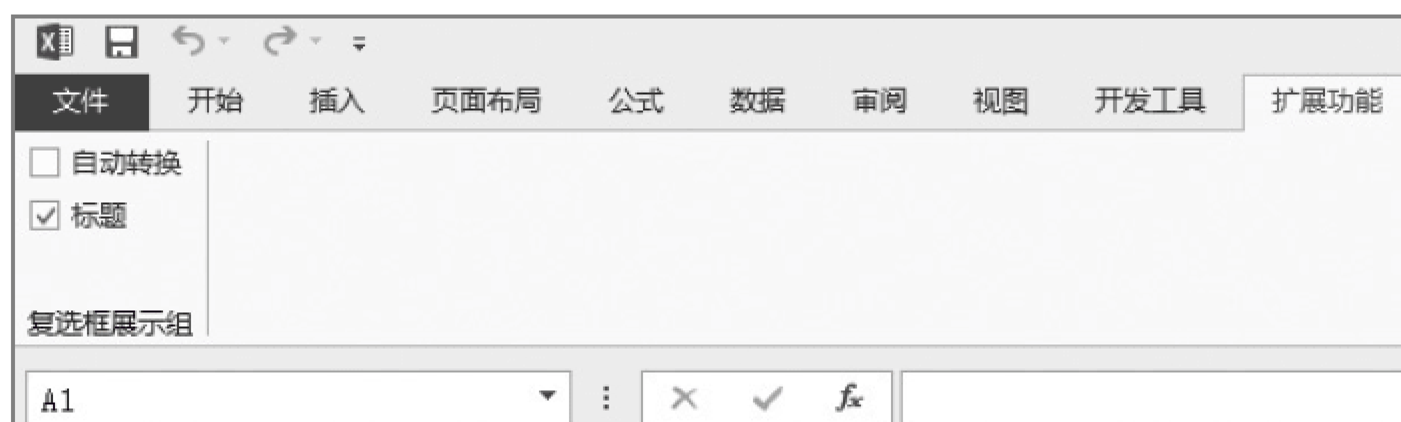


图 5-29　添加自定义复选框

注意　上面的 XML 代码不能用于 Excel 以外的 Office 组件，因为 idMso="ViewHeadings" 这个内置控件是 Excel 独有的。

5.3.6　组合框

组合框（comboBox）是一种容器控件，其中的条目由 item 元素来规定。下面的 XML 在 group 中加入了一个内置的字体组合框，并且创建了一个用户自定义组合框。自定义组合框中的内容是 12 个月份的英文。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
```

```
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 组合框展示组 ">
                    <comboBox idMso="Font"/>
                    <comboBox id="Combo1" label=" 月份 ">
                        <item id="Month1" label="January"/>
                        <item id="Month2" label="February"/>
                        <item id="Month3" label="March"/>
                        <item id="Month4" label="April"/>
                        <item id="Month5" label="May"/>
                        <item id="Month6" label="June"/>
                        <item id="Month7" label="July"/>
                        <item id="Month8" label="August"/>
                        <item id="Month9" label="September"/>
                        <item id="Month10" label="October"/>
                        <item id="Month11" label="November"/>
                        <item id="Month12" label="December"/>
                    </comboBox>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 代码对应的 Excel 界面如图 5-30 所示。

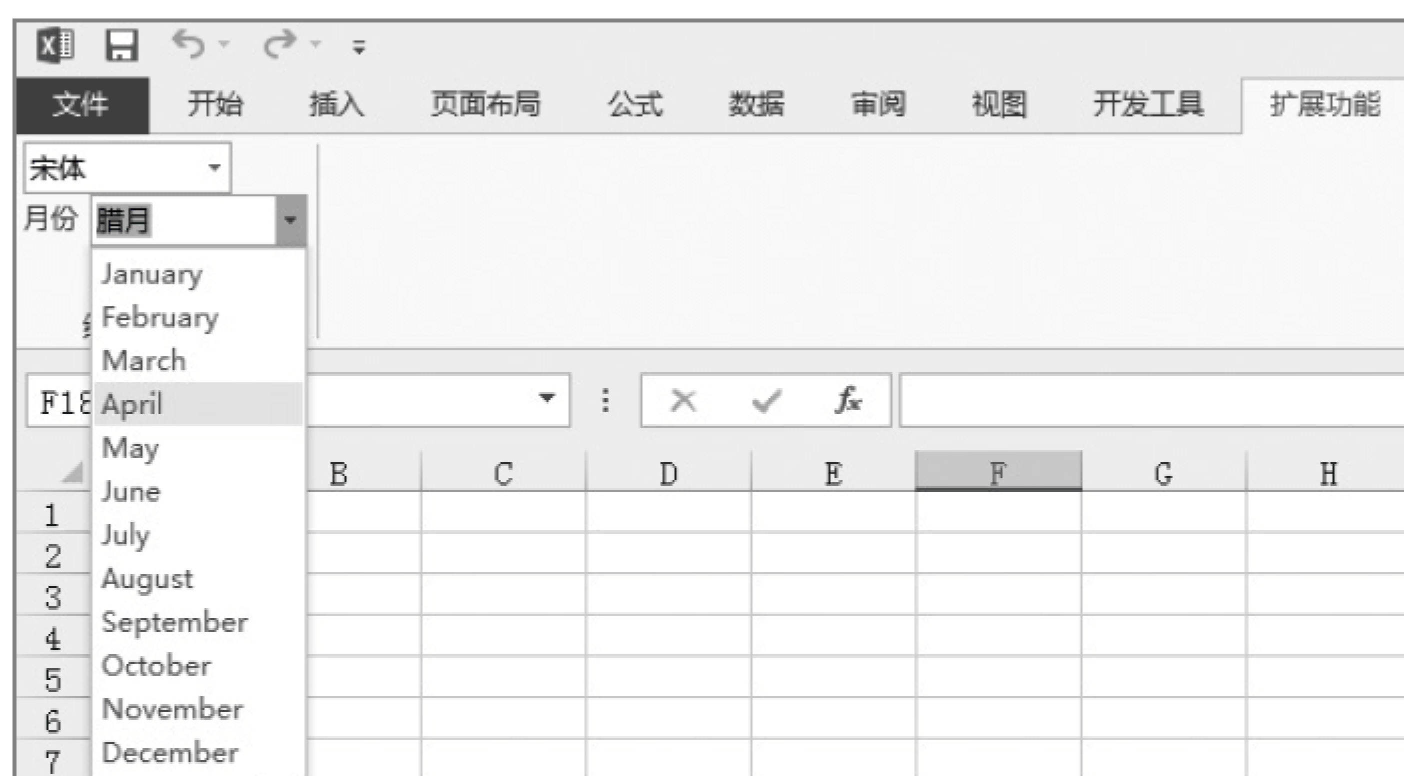


图 5-30　添加自定义组合框

组合框的取值，既可以来自于任意一个 item 的 label，同时也可以像文本框一样接受用户输入，如图 5-30 所示，可以用键盘输入“腊月”两个字。

5.3.7　下拉框

下拉框（dropDown）与组合框的主要区别是，下拉框不接受用户键盘输入，只能从下拉条目中选择。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
```

```
                <group id="Group2019" label=" 下拉框展示组 ">
                    <dropDown id="Drop1" label=" 月份 ">
                        <item id="Month1" label="January"/>
                        <item id="Month2" label="February"/>
                        <item id="Month3" label="March"/>
                    </dropDown>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-31 所示。

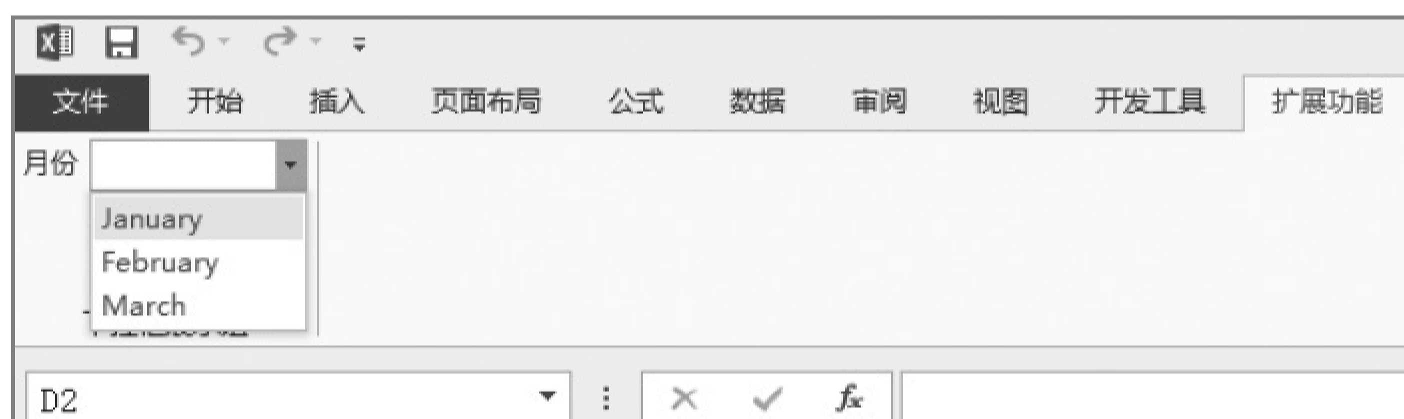


图 5-31　添加自定义下拉框

5.3.8　文本框

文本框（editBox）是应用非常广泛的一类控件，可以让用户输入一些文本、数字等内容，如果要显示文本框左侧的说明性文字，需要设置 showLabel 属性为 true ；设置为 false 则隐藏标签文字。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 文本框展示组 ">
                    <editBox id="EditBox1" showLabel="true" label=" 姓名 :"/>
                    <editBox id="EditBox2" showLabel="true" label=" 年龄 :"/>
                </group>
            </tab>
            </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-32 所示。

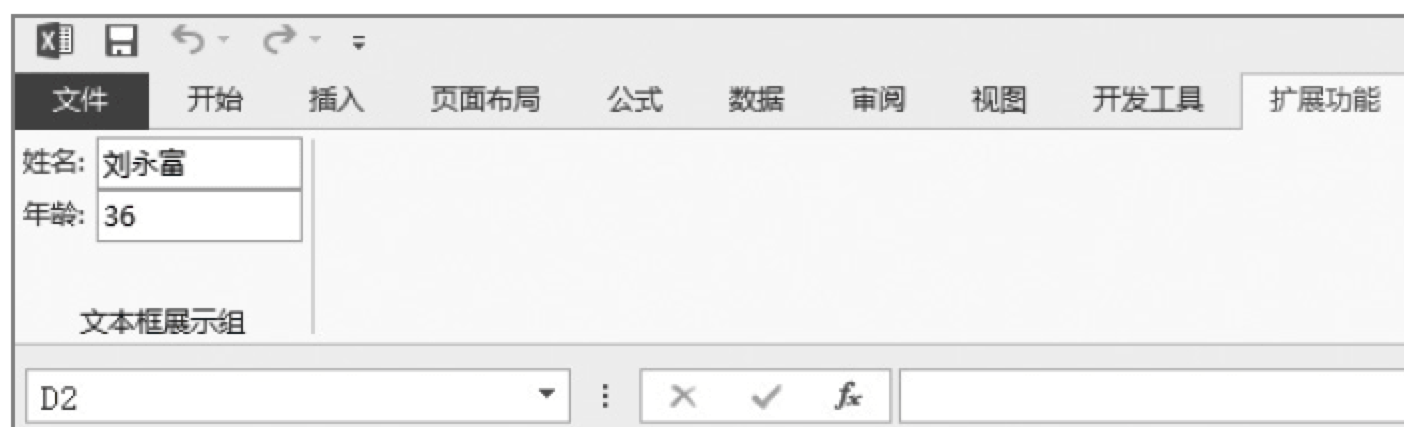


图 5-32　添加自定义文本框

5.3.9 标签

标签（labelControl）控件用于显示一些说明文字，规定其 label 属性即可。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 标签展示组 ">
                    <labelControl id="Label1" label=" 更多选项 "/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-33 所示。

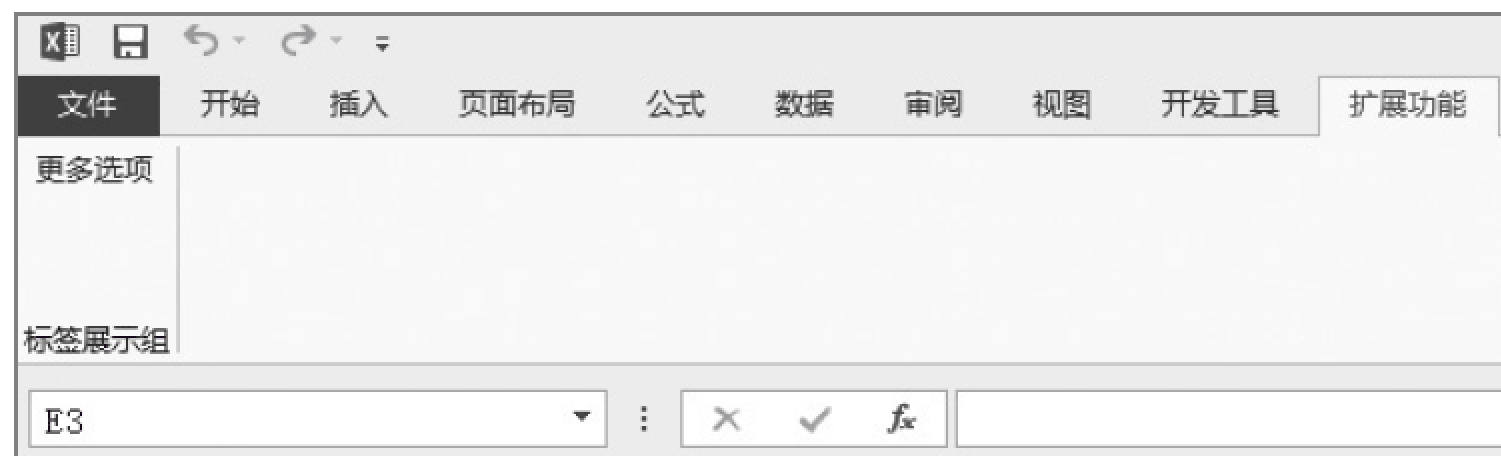


图 5-33　添加自定义标签

5.3.10 分隔线

分隔线（separator）用于隔离相邻的控件，正常的控件一般是上下方向可以放置 3 个控件，但是插入 separator 后，后续的控件出现在下一列。

下面的 XML 首先放入一个标签控件，然后加入分隔线，再放入另外一个标签控件。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 分隔线展示组 ">
                    <labelControl id="Label1" label=" 更多选项 "/>
                    <separator id="Separator1"/>
                    <labelControl id="Label2" label=" 功能定制 "/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-34 所示。

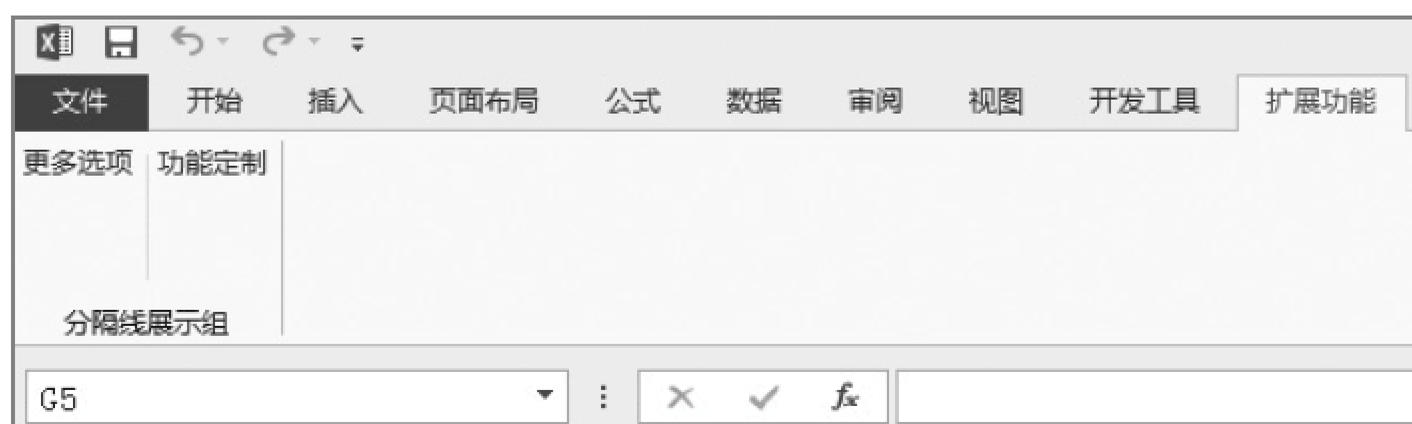


图 5-34　添加分隔线

5.3.11　切换按钮

切换按钮（toggleButton）有按下和弹起两种状态，例如字体的加粗、倾斜都是这类控件。下面的 XML 向 group 中添加一个自定义切换按钮，然后添加一个内置的加粗切换按钮。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 切换按钮 ">
                    <toggleButton id="Toggle1" label=" 斜体 "/>
                    <toggleButton idMso="Bold"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML对应的 Excel 界面如图 5-35 所示。

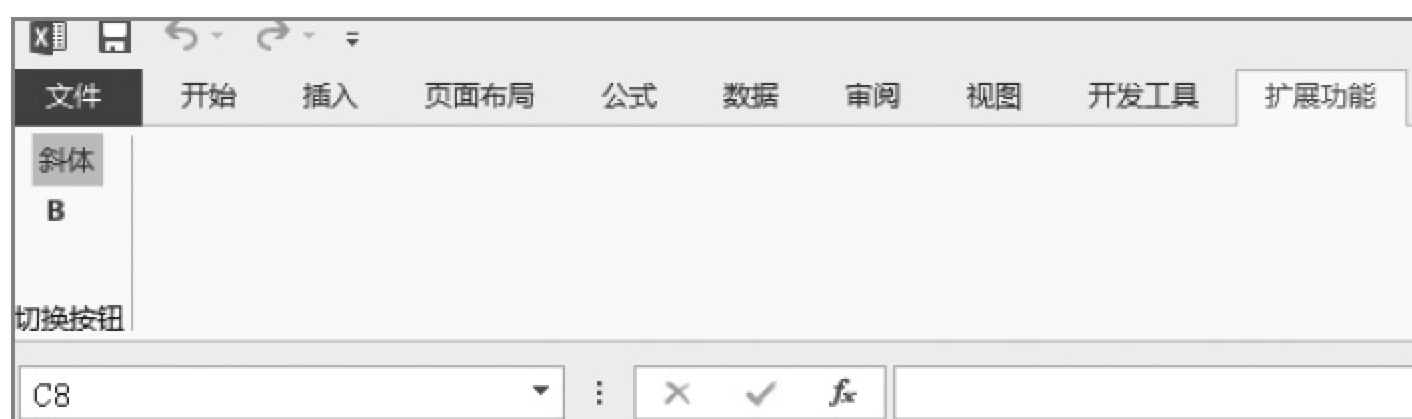


图 5-35　添加自定义切换按钮

5.3.12　控件箱

控件箱（box）是一个容器控件，该容器可以把多个控件捆绑为一体。box 的 boxStyle 属性有 horizontal 和 vertical 两种取值。

下面的 XML 分别用水平捆绑、垂直捆绑 Button1 和 Check1 控件。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 水平捆绑 ">
                    <box id="Box1" boxStyle="horizontal">
```

```
                <button id="Button1" label="Button1"/>
                <checkBox id="Check1" label="Check1"/>
            </box>
            <editBox id="Edit1" label="Name:"/>
        </group>
        <group id="Group2020" label=" 垂直捆绑 ">
            <box id="Box2" boxStyle="vertical">
                <button id="Button2" label="Button1"/>
                <checkBox id="Check2" label="Check1"/>
            </box>
            <editBox id="Edit2" label="Name:"/>
        </group>
        <group id="Group2021" label=" 一般显示 ">
            <button id="Button3" label="Button1"/>
            <checkBox id="Check3" label="Check1"/>
            <editBox id="Edit3" label="Name:"/>
        </group>
    </tab>
  </tabs>
 </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-36 所示。

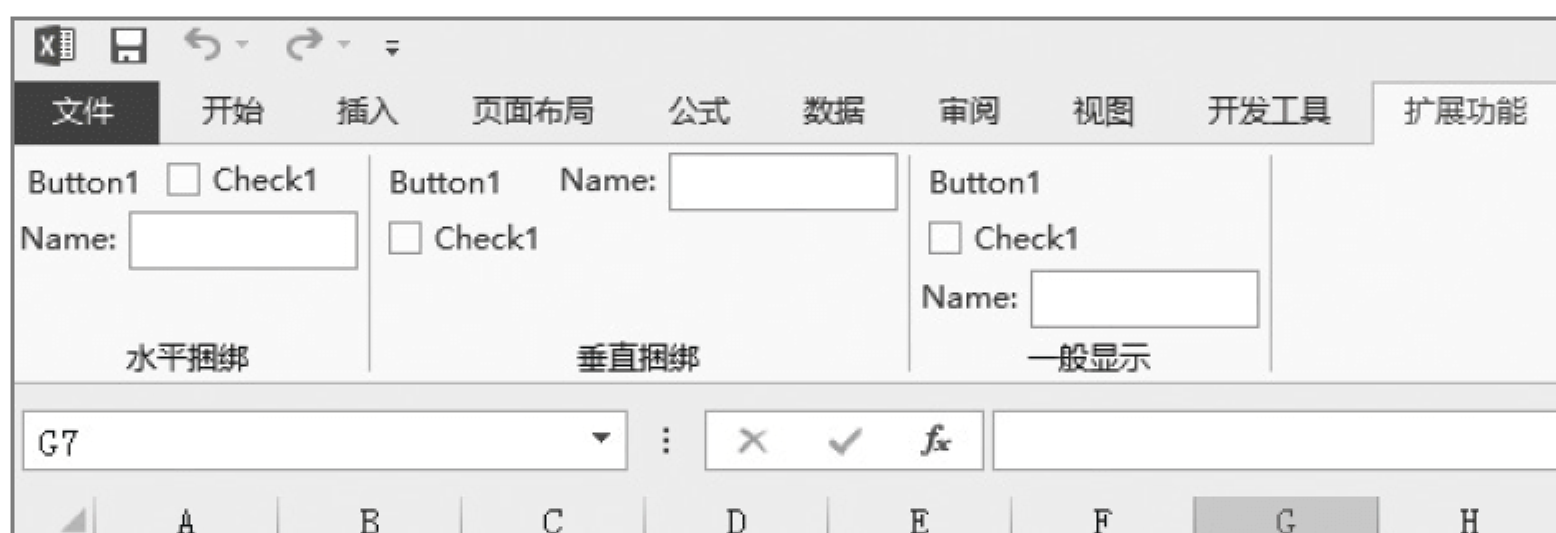


图 5-36 使用控件箱

可以看出 box本身没有任何外观显示，在“水平捆绑”组中，Button1 和 Check1 进行水平捆绑，因此这两个控件显示在同一行，其余控件另起一行。

在“垂直捆绑”组中，被捆绑的控件单独占据一列，其余控件必须另起一列。

在不使用 box 的“一般显示”组中，3 个控件上下排列在同一列。

5.3.13 控件组

控件组（buttonGroup）的功能与 box 非常相似，但与 box 有好几处不同，如下所示。

- ❑ 只能平捆绑。
- ❑ 可以包含的子控件类型有限。

下面的 XML 首先向 group 中添加一个 buttonGroup 控件，其次往该控件组中添加 3 个子控件。最后向 group 中添加一个文本框。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
```

```
        <tabs>
            <tab id="Tab2018" label="扩展功能">
                <group id="Group2019" label="水平捆绑">
                    <buttonGroup id="ButtonGroup1">
                        <button id="Button1" label="Button1"/>
                        <toggleButton id="Toggle1" label="Toggle1"/>
                        <control idMso="FormatPainter"/>
                    </buttonGroup>
                    <editBox id="Edit1" label="Name:"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-37 所示。

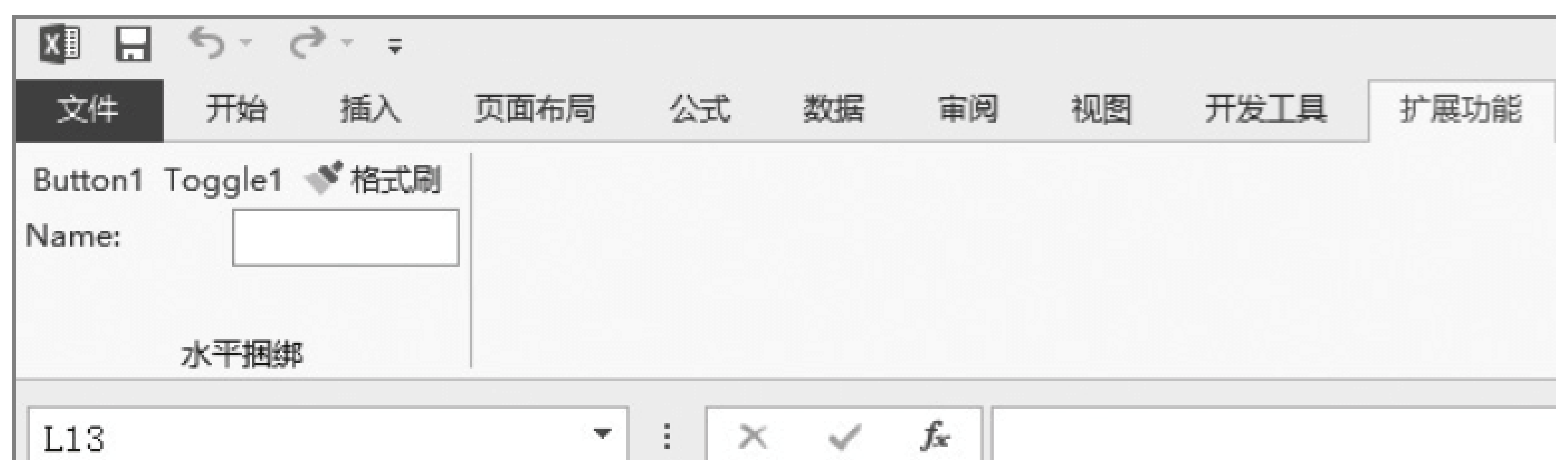


图 5-37　使用控件组

由于按钮、切换按钮、格式刷三个控件被水平放入控件组中，因此不能换行，只能保持在同一行。

5.3.14　图片库

图片库（gallery）是一种容器控件，下面可以容纳 item 元素。gallery 本身也是一种控件，可以设置其 image、label 属性。更重要的是要设置其子元素的行数和列数。

下面的 XML 代码在 gallery 中呈现 3 行 2 列的 item。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label="扩展功能">
            <group id="Group2019" label="3 行 2 列">
                    <gallery id="Gallery1" label="我的图库" imageMso="G" rows="3"
columns="2">
                        <item id="pic1" label="pic1" imageMso="A"/>
                        <item id="pic2" label="pic2" imageMso="B"/>
                        <item id="pic3" label="pic3" imageMso="C"/>
                        <item id="pic4" label="pic4" imageMso="D"/>
                        <item id="pic5" label="pic5" imageMso="E"/>
                        <item id="pic6" label="pic6" imageMso="F"/>
                    </gallery>
                </group>
            </tab>
        </tabs>
```

```
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-38 所示。

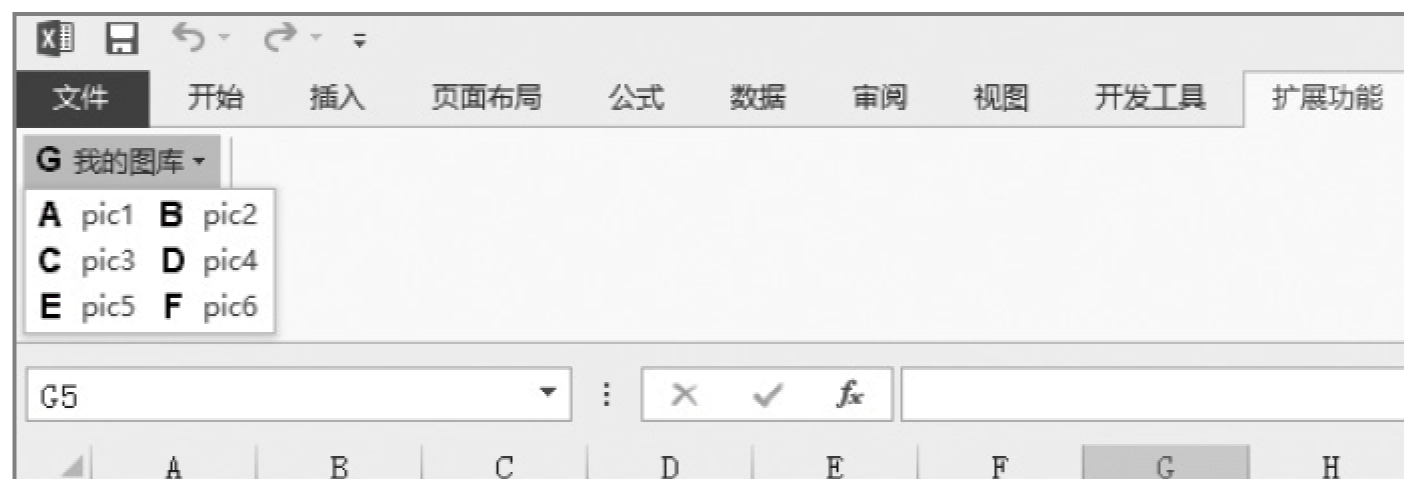


图 5-38 使用图片库

如果更改 rows="3" columns="2" 为 rows="2" columns="3"，则效果如图 5-39 所示。

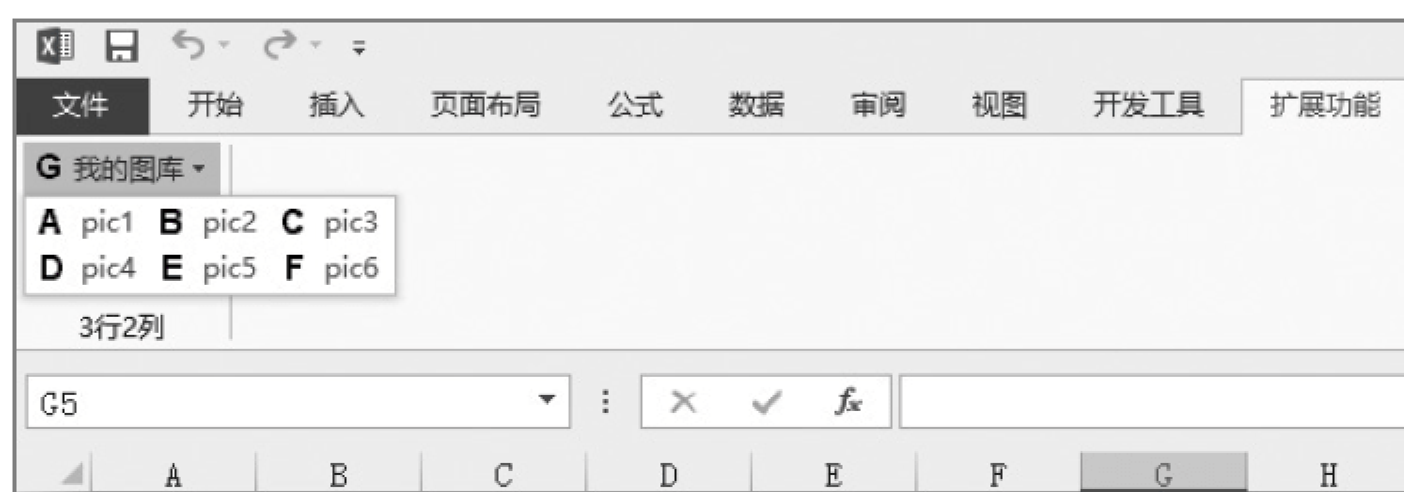


图 5-39 更改图片库的行列数

此外，还可以设置 itemWidth 和 itemHeight 来更改子元素的尺寸。更改 gallery 的属性为如下形式。

```
<gallery id="Gallery1" label=" 我的图库 " imageMso="G" rows="3" columns="2" size=
"large" itemWidth="30" itemHeight="30">
```

与上述 XML 对应的 Excel 界面如图 5-40 所示。

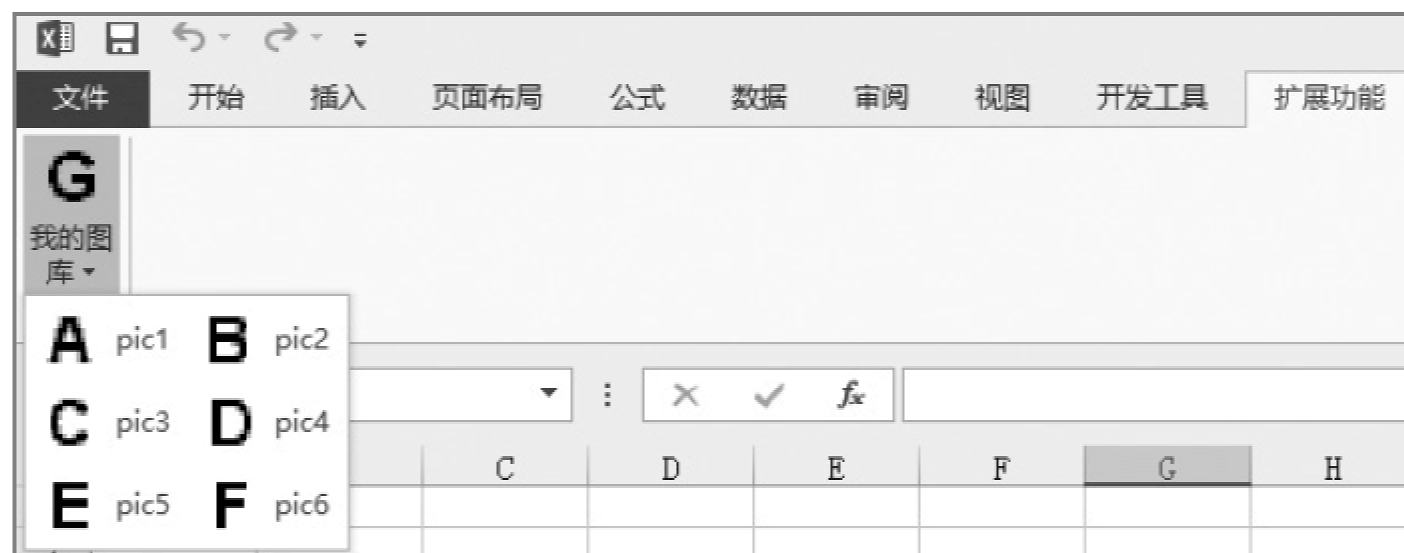


图 5-40 更改项目的宽度、高度

gallery 除了显示 imageMso 指代的内置图像外，还可以显示计算机中的自定义图片。

5.3.15 菜单

菜单（menu）元素也是一种容器控件，下面可以放置 button、checkBox 等简单控件，也可以使用 menuSeparator 作为菜单项的分隔线，还可以在 menu 之下再放置 menu 元素作为子菜单。

下面的 XML 在 group 下面添加一个 menu，menu 下面放置一个 button，然后放置一个菜单分隔线，再放置一个按钮。

接下来在 menu 中再创建一个 menu 作为子菜单，该子菜单下面放置两个 checkBox。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 菜单控件 ">
                    <menu id="Menu1" label=" 东北地区 ">
                        <button id="Button1" label=" 黑龙江省 "/>
                        <menuSeparator id="Separator1" />
                        <button id="Button2" label=" 吉林省 "/>
                        <menu id="Menu2" label=" 辽宁省 ">
                            <checkBox id="Check1" label=" 沈阳市 "/>
                            <checkBox id="Check2" label=" 大连市 "/>
                        </menu>
                    </menu>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-41 所示。

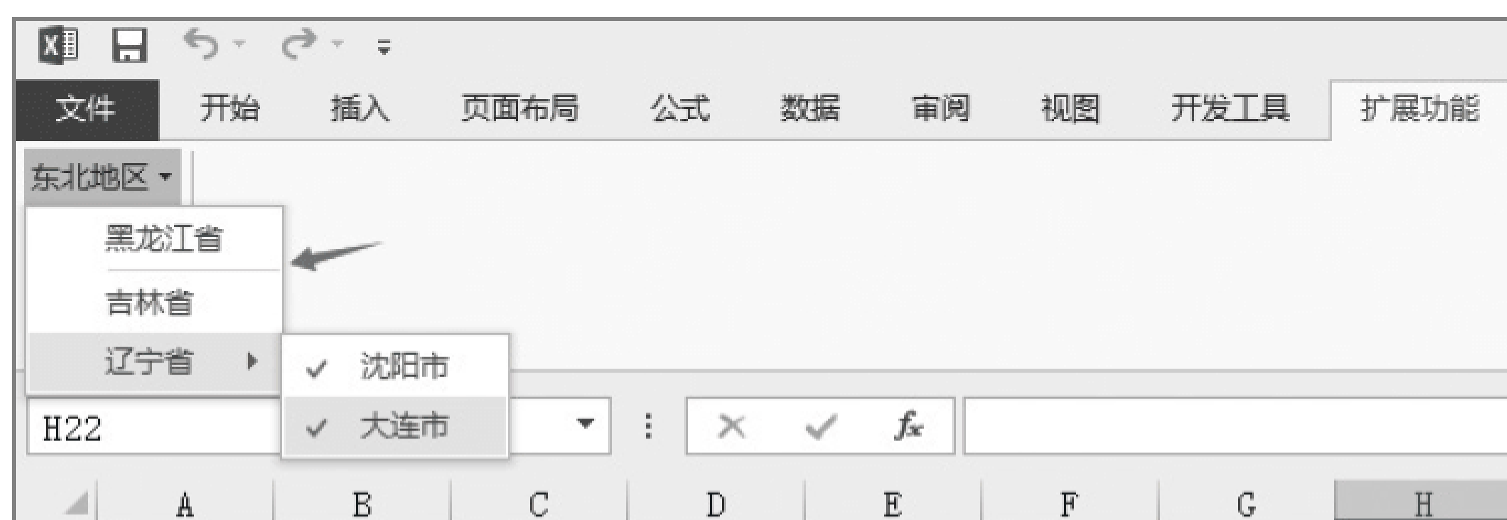


图 5-41　添加自定义级联菜单

可以看到“吉林省”这个按钮上方有一分隔线。

5.3.16　分裂按钮

分裂按钮（splitButton）与 menu 相似，但是比 menu 稍稍复杂。可以把 splitButton 理解为一个 button 和一个 menu 的组合体。例如【开始 / 剪贴板】中的“复制”就是一个 splitButton，既可以直接单击该控件的主体按钮部分，也可以单击右侧小箭头，随后可以弹出子菜单，如图 5-42 所示。

splitButton 的 XML 结构如图 5-43 所示。

下面的 XML 创建了一个 splitButton，然后添加一个“常规早餐”的按钮，在按钮下面增加一个菜单，该菜单中包含两个复选框。

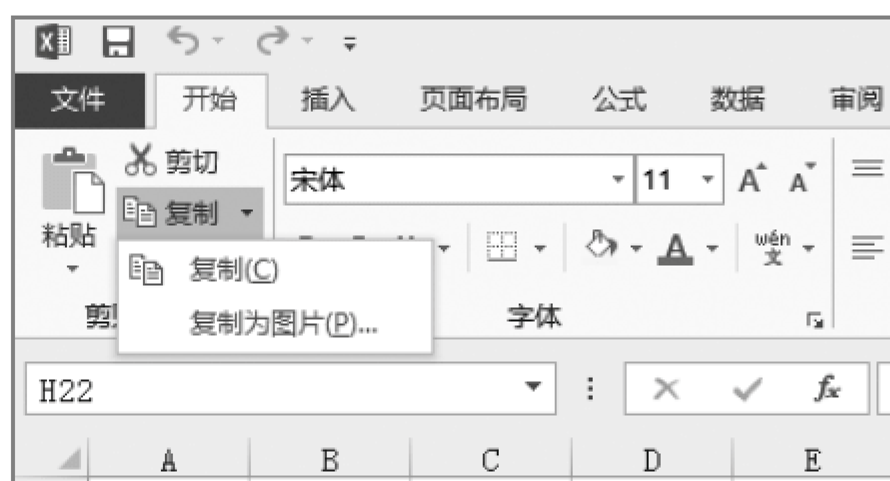


图 5-42 内置分裂按钮

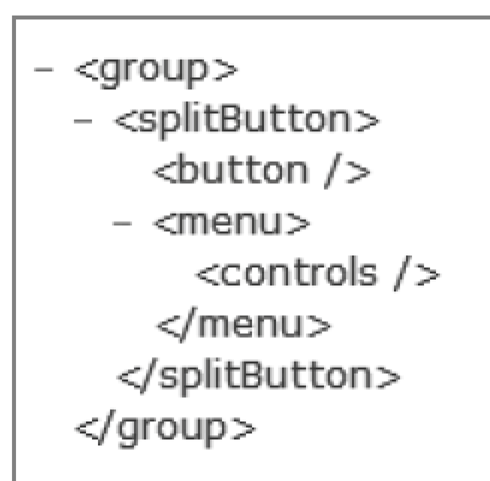


图 5-43 splitButton 的 XML 结构

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 分裂菜单控件 ">
                    <splitButton id="Split1">
                        <button id="Button1" label=" 常规早餐 "/>
                        <menu id="Menu1">
                            <checkBox id="Check1" label=" 加牛奶 "/>
                            <checkBox id="Check2" label=" 加鸡蛋 "/>
                        </menu>
                    </splitButton>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-44 所示。

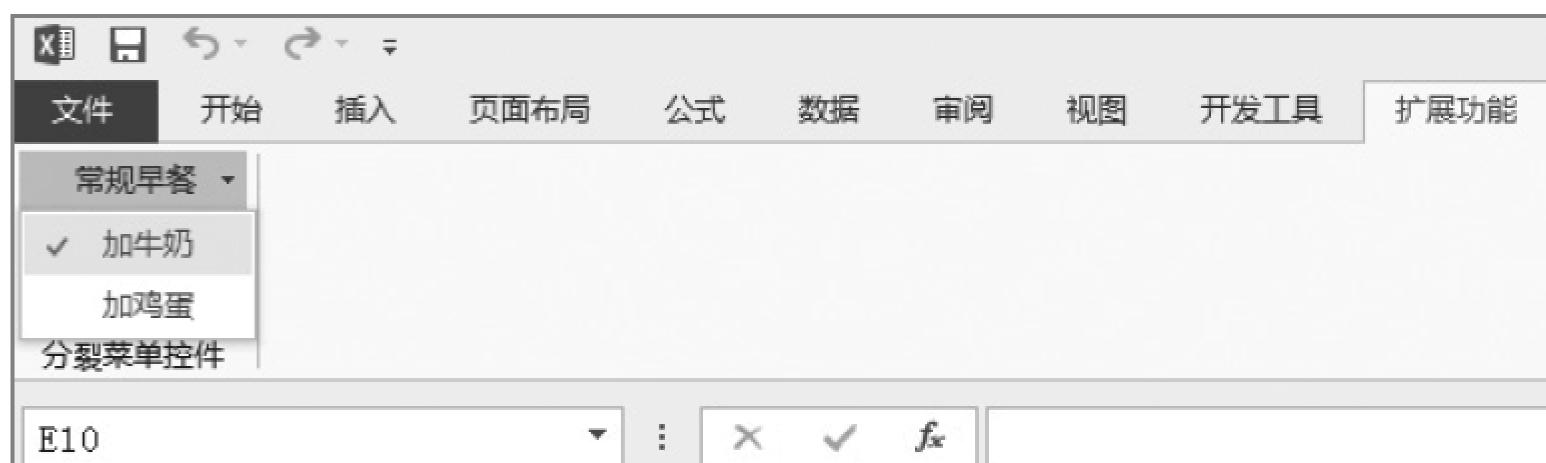


图 5-44 添加自定义分裂按钮

5.3.17 动态菜单

动态菜单（dynamicMenu）和 menu 在外观上都是菜单，但是实现原理非常不同，前面讲过，制作 menu 时，需要把 menu 下面包含的控件以 XML 代码的形式包含在 menu 元素之下。而 dynamicMenu 则是 customUI 加载后，后期可以增删菜单中的项目。

dynamicMenu 控件必不可少的回调是 getContent，而 getContent 对应的回调函数必须返回一段以 <menu> 为根节点的 XML 代码方可。

“实例文档 28.xlsm”的 customUI 的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="customUI_Load">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能">
                <group id="Group1" label="动态菜单">
                    <dynamicMenu id="Dynamic1" label="城市名称" getContent= "GetXML"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

可以看到，group 下面只有一个 dynamicMenu 控件，其 getContent 回调函数是 GetXML 函数，因此 VBA 代码如下。

```
'添加外部引用 Microsoft XML v6.0
Public R As Office.IRibbonUI
Public Sub customUI_Load(ribbon As Office.IRibbonUI)
    Set R = ribbon
End Sub

Public Sub GetXML(control As Office.IRibbonControl, ByRef content)
    Dim x As New MSXML2.DOMDocument, Root As MSXML2.IXMLDOMElement
    Dim E As MSXML2.IXMLDOMElement, i As Integer
    With x
        Set Root = x.createElement("menu")
        Root.setAttribute "xmlns", "http://schemas.microsoft.com/office/2009/07/
customui"
        Set x.DocumentElement = Root
        For i = 2 To Sheet1.Range("A1").End(xlDown).Row
            Set E = x.createElement(Sheet1.Range("A" & i).Value)
            With E
                .setAttribute "id", Sheet1.Range("B" & i).Value
                .setAttribute "label", Sheet1.Range("C" & i).Value
                .setAttribute "tag", Sheet1.Range("D" & i).Value
                .setAttribute "onAction", Sheet1.Range("E" & i).Value
            End With
            Root.appendChild E
        Next i
        content = x.XML
    End With
End Sub
Public Sub OA(control As Office.IRibbonControl)
    MsgBox control.Tag
End Sub
Sub Update()
    R.InvalidateControl "Dynamic1"
End Sub
```

代码分析：GetXML 函数使用第 4 章讲过的 VBA 自动生成 XML 的方式，从工作表单元格中的预先设定数据生成 XML 字符串，最后赋给 GetXML 函数。

打开该工作簿，编辑工作表中的数据，单击工作表上的“更新菜单”按钮，会看到城市名称菜单中的项目会随之变化，如图 5-45 所示。

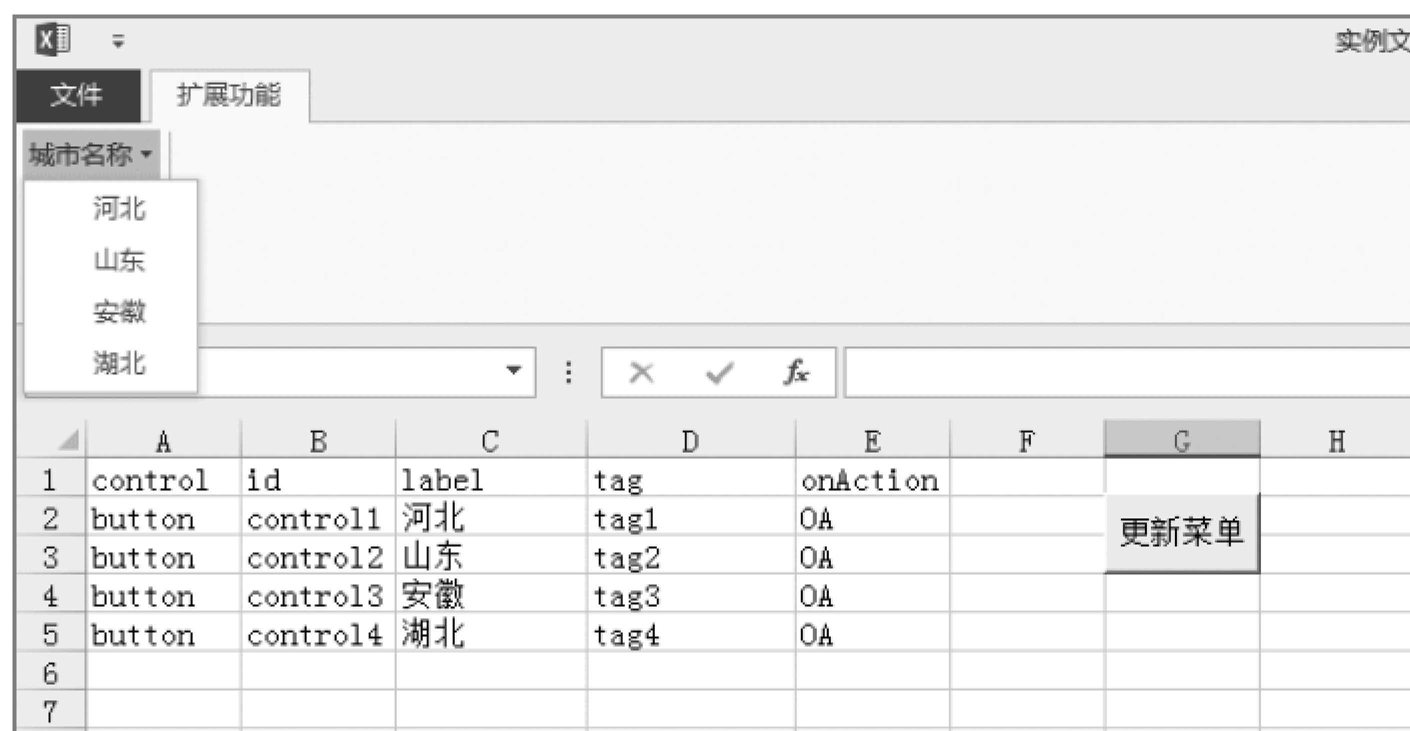


图 5-45　动态菜单

关于如何刷新功能区，可以参考 5.4.10 节和 5.4.11 节。

5.3.18　对话框

一般情况下，button 直属于 group 之下，并且每个 button 都会占据 group 中的一个位置。使用对话框（dialogBoxLauncher）元素可以把一个 button 置于组的右下角，以小箭头呈现。当用户单击右下角的小箭头时，就相当于单击了这个 button，也就是说 dialogBoxLauncher 元素本身没有任何回调。

dialogBoxLauncher 的使用非常简单，它的父级元素必须是 group，它的子元素必须是 button，而且只能是一个 button。而且一个 group 之下最多允许一个 dialogBoxLauncher。

下面的 XML 代码，在 group 之下放置一个 dialogBoxLauncher 元素，然后放置一个 button。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab2018" label=" 扩展功能 ">
                <group id="Group2019" label=" 对话框 ">
                    <dialogBoxLauncher>
                        <button id="Button1" onAction="ShowMyForm" />
                    </dialogBoxLauncher>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-46 所示。

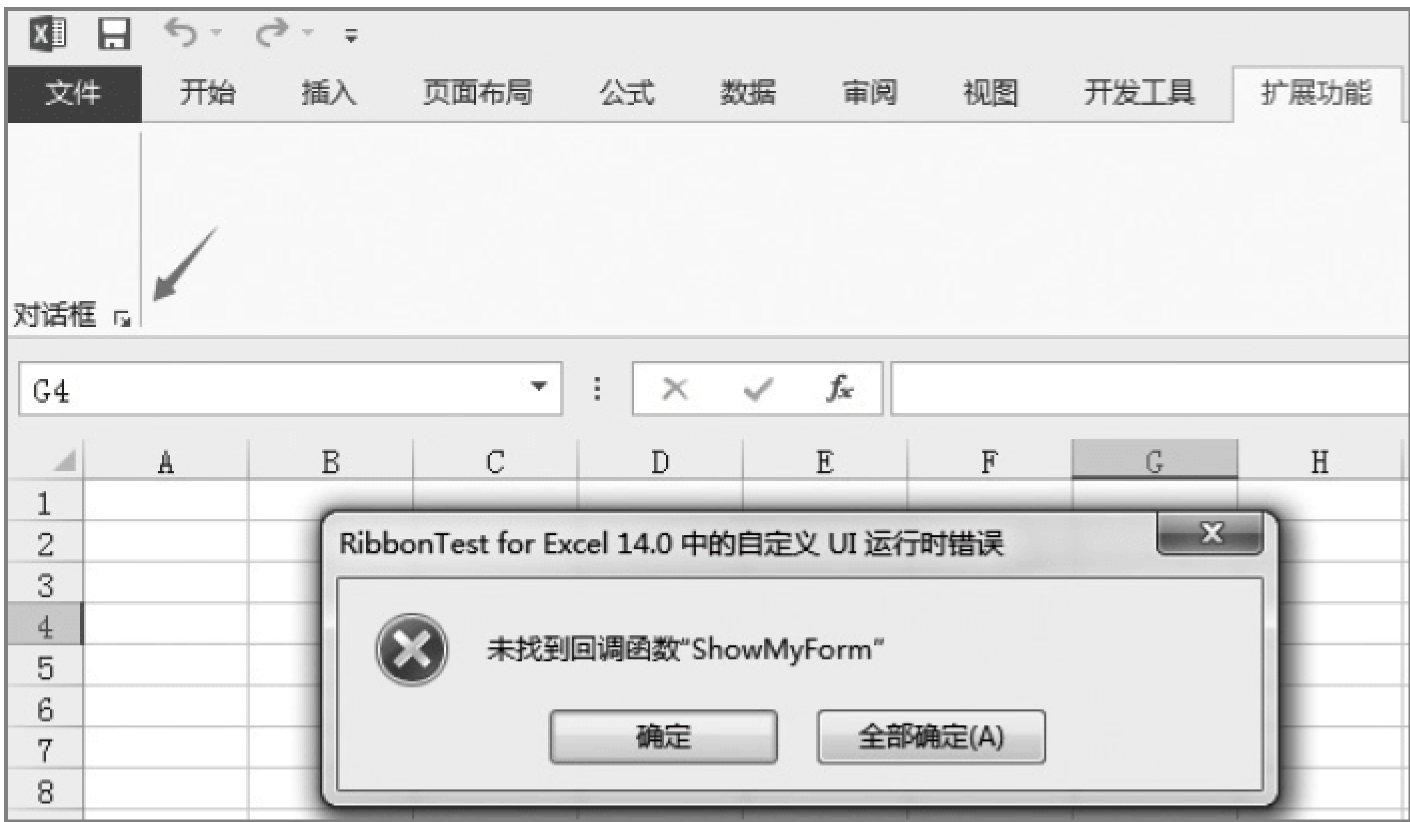


图 5-46　组右下角的对话框控件

当单击右下角的小箭头后，正常情况下会调用 VBA 中的 ShowMyForm 过程。本例由于没有书写相应的回调，弹出了“自定义 UI 运行时错误”，这属于正常现象。

5.4　常用属性详解

用于自定义功能区的 XML 代码中，每个元素都可以设置诸多属性来控制控件的实际显示风格。因此，除了前面介绍过的 id 和 label 属性之外，还需要了解更多重要属性，如表 5-2 所示。

表 5-2　customUI 属性及含义（按字母排序）

属　　性	类型或取值	描　　述
description	文本	当 menu 中的 itemSize 属性设置为 large 时显示的描述文本
enabled	true, false, 0, 1	规定控件是否可用
getContent	回调	只用于动态菜单（dynamicMenu），设置返回的动态 XML 代码
getDescription	回调	动态设置控件的描述文本
getEnabled	回调	动态设置控件的可用性
getImage	回调	动态设置控件的图标
getImageMso	回调	动态设置控件的内置图标
getItemCount	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项总数
getItemID	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项的 id
getItemImage	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项的图标
getItemLabel	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项的标题
getItemScreentip	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项的 screenTip
getItemSupertip	回调	动态设置容器控件（comboBox、dropDown、gallery）包含的子项的 superTip
getKeytip	回调	动态设置控件的 KeyTip
getLabel	回调	动态设置控件的标题文字

续表

属　　性	类型或取值	描　　述
getPressed	回调	动态设置 checkBox、toggleButton 控件的按下状态
getScreentip	回调	动态设置控件的 screenTip 属性
getSelectedItemID	回调	对于 dropDown 或 gallery 控件，动态设置所选条目的 id
getSelectedItemIndex	回调	对于 dropDown 或 gallery 控件，动态设置所选条目的序号
getShowImage	回调	动态设置是否显示图标
getShowLabel	回调	动态设置是否显示标题文字
getSize	回调	动态设置控件的尺寸，是 normal 还是 large
getSupertip	回调	动态设置控件的 superTip 属性
getText	回调	动态设置 editBox 控件的文本
getTitle	回调	对于 menu 中的 menuSeparator 控件，动态设置其标题
getVisible	回调	动态设置控件的可见性
id	文本	用户自定义元素的唯一识别号，不可与 idMso、idQ 同时使用
idMso	控件编号	内置控件的唯一识别号
idQ	文本	指定命名空间的控件识别号
image	文本	规定控件的图标
imageMso	控件编号	规定控件显示内置图标
insertAfterMso	控件编号	规定置于哪一个内置控件之后
insertBeforeMso	控件编号	规定置于哪一个内置控件之前
itemSize	large,normal	对于 menu，规定其子项的尺寸
keytip	文本	规定控件的快捷键，按下 Alt 键显示快捷键提示
label	文本	规定控件的标题
onAction	回调	当单击控件时触发的过程
onChange	回调	当修改 comboBox、editBox 内容时触发的过程
screentip	文本	规定控件的 screenTip
showImage	true，false，0，1	规定是否显示图标
showItemImage	true，false，0，1	用于 comboBox、dropDown、gallery 控件，规定是否显示子项的图标
showItemLabel	true，false，0，1	用于 comboBox、dropDown、gallery 控件，规定是否显示子项的标题
showLabel	true，false，0，1	规定是否显示控件的标题
size	large，normal	规定控件的尺寸
sizeString	文本	暗示 editBox 的宽度
supertip	文本	规定控件的 superTip 属性
tag	文本	用户自定义文本，一般用来区分不同的控件
title	文本	用于 menu 中的 menuSeparator 控件，规定其标题
visible	true，false，0，1	规定控件的可见性

属性从控件来源可以划分为自定义属性和内置属性。例如规定一个控件的 id 属性，说明该控件是自定义的；如果规定一个控件的 imageMso 属性，说明该图像来源于 Office 内置图标。

属性从作用原理上可以划分为直接属性和回调属性。例如控件的 label 属性就是一次

性规定这个控件的标题文字，后期不得更改。而规定控件的 getLabel 属性，就意味着该控件的标题文字可以由 VBA 代码产生。很多以 get 开头或者 on 开头的属性都是回调属性，也就是说，如果在 XML 代码中使用了回调属性，那么在 VBA 代码中一定要有相应的回调函数来呼应。

下面把相似的属性放在一起对比讲解。

5.4.1　id-idMso

id 属性是自定义元素的唯一识别符，tab 元素及其以下几乎都需要规定 id 值，也有个别的不需要 id，例如 dialogBoxLauncher。

idMso 的作用是引用 Office 内置元素。

Office 2013 常用组件的 idMso 可以使用笔者制作的 OfficeidMsoViewer 软件来快速查询和获取，如图 5-47 所示。

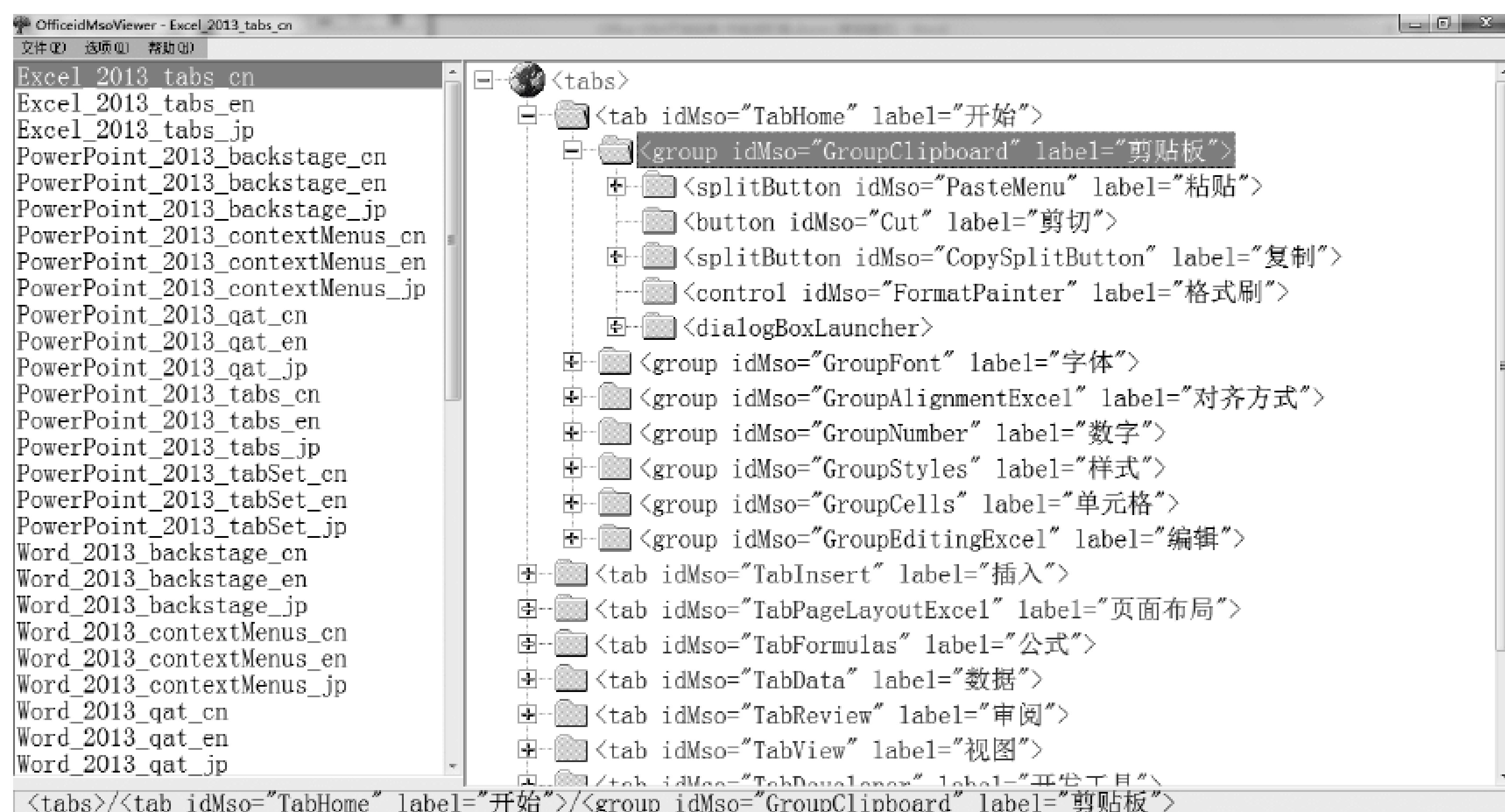


图 5-47　OfficeidMsoViewer 界面

单击右侧树状结构中任意一个节点时，会立即生成该节点对应的 XML 代码。

例如，在新的自定义 tab 中引用内置的 GroupFont、GroupNumber 和 GroupCells，可以使用如下 XML 代码。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group idMso="GroupFont">
                </group>
                <group idMso="GroupNumber">
                </group>
                <group idMso="GroupCells">
                </group>
            </tab>
```

```
        </tabs>
    </ribbon>
</customUI>
```

可以看到以上 3 个内置组出现在“扩展功能”自定义选项卡中，如图 5-48 所示。

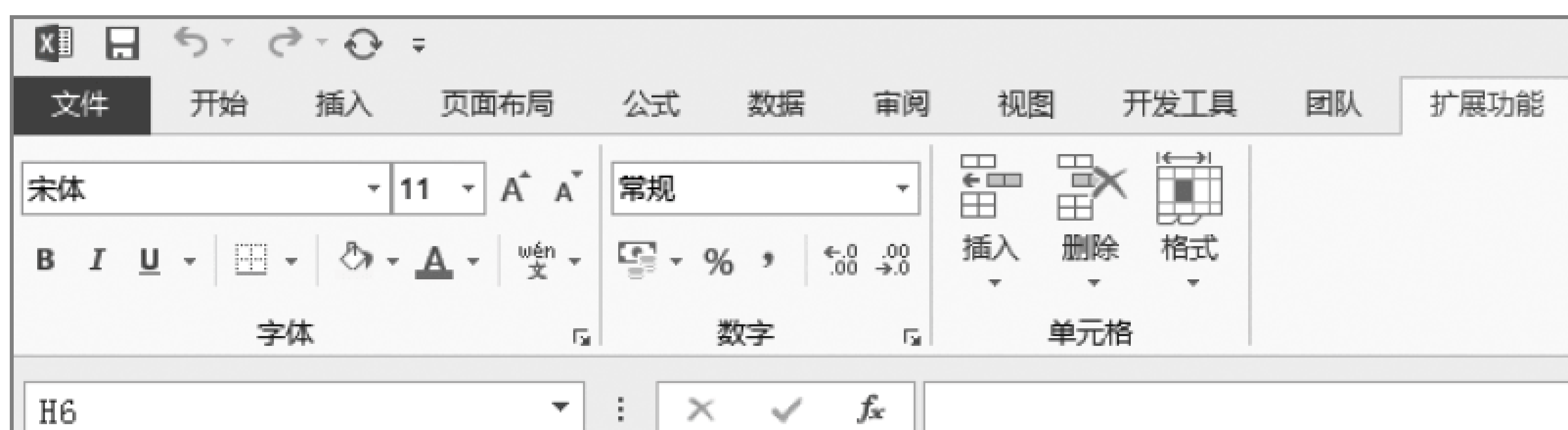


图 5-48 自定义选项卡中引用内置组

同一个元素不能同时使用 id 和 idMso 属性。

5.4.2 insertBeforeMso-InsertAfterMso

InsertBeforeMso 是指该元素位于指定的内置元素之前。insetAfterMso 是之后。

例如，<tab id="Tab1" label=" 扩展功能 " insertAfterMso="TabHome"> 是指创建一个新选项卡，该选项卡处于内置选项卡“开始”的后面。

下面的 XML 代码能在“开始”选项卡之后新建一个自定义空白选项卡。然后在“开始”内置选项卡的“数字”内置组之前创建一个自定义组。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 " insertAfterMso="TabHome">
            </tab>
            <tab idMso="TabHome">
                <group id="Group1" insertBeforeMso="GroupNumber" label=" 自定义组 ">
                    <button id="Button1" label=" 自定义按钮 "/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-49 所示。



图 5-49 前后位置调整

5.4.3　enabled-getEnabled

enabled 属性指的是元素的有效性，取值可以是 true（1）或者 false（0）。

getEnabled 是回调属性，元素的有效性取决于 VBA 返回的结果。

例如，在 XML 中规定两个 button，设置不同的 enabled 属性。

```
<button id="Button1" label="自动转换" enabled="false"/>
<button id="Button2" label="手工转换" enabled="1"/>
```

Excel 中可以看到“自动转换”这个 button 是灰色不可用的，如图 5-50 所示。

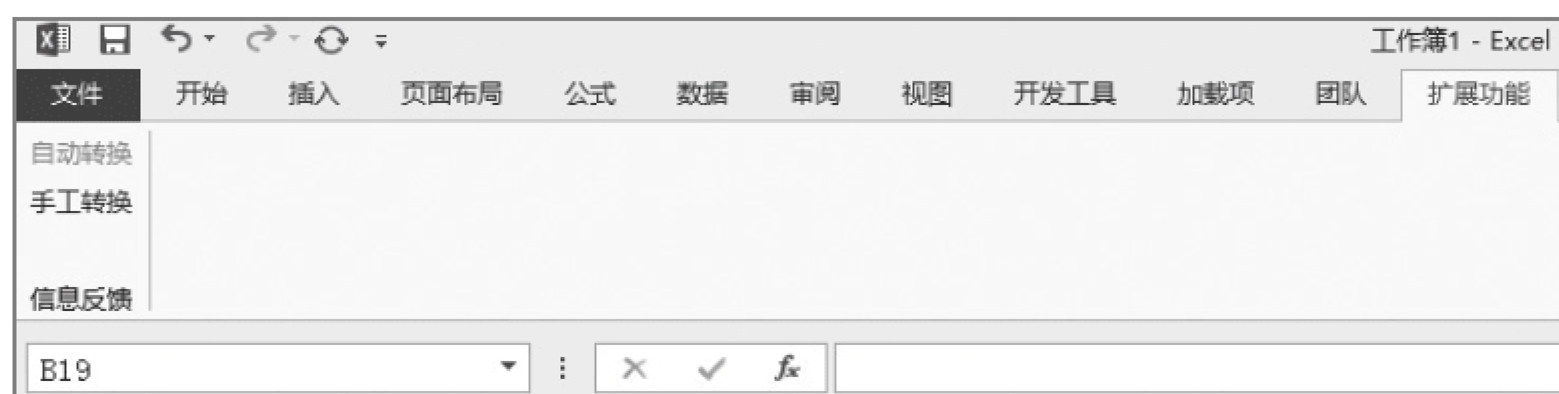


图 5-50　控件的可用性

使用 getEnabled 属性可以智能地由 VBA 过程来决定控件是否可用。

“实例文档 16.xlsm”中包含了下面的 XML 代码和 VBA 代码，用于演示 getEnabled 功能。

XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能">
                <group id="Group1" label="高级技术">
                      <button id="Button1" label="自动转换" getEnabled="Button_
Enabled"/>
                      <checkBox id="Check1" label="递归算法" getEnabled="Check_
Enabled"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

以上两个控件相应的 VBA 回调如下。

```
Public Sub Button_Enabled(control As Office.IRibbonControl, ByRef enabled)
    enabled = True
End Sub
Public Sub Check_Enabled(control As Office.IRibbonControl, ByRef enabled)
    Dim flag As Boolean
    Randomize
    flag = Rnd() > 0.5
    enabled = flag
End Sub
```

代码分析：回调函数中的 ByRef enabled 是一个布尔值，用于决定控件的可用性，因此回调函数中要更改该参数的取值。

本例中由于按钮的 enabled=True，复选框的 enabled 取决于 flag 的值，而 flag 是用随机数和 0.5 比较大小，因此复选框有两种随机状态：可用或不可用。

在 Excel 中打开该文档，加载 customUI 的同时，会自动运行标准模块中的相关回调函数，Excel 中的效果如图 5-51 所示。

当关闭该文档，再次打开时，复选框或许会变得可用。

从以上实例可以看出控件的 enabled 属性不依赖于 VBA 中的回调，而是在 XML 直接指定控件是否可用。而 getEnabled 属性则需要根据 VBA 过程中参数的返回值来决定控件是否可用。

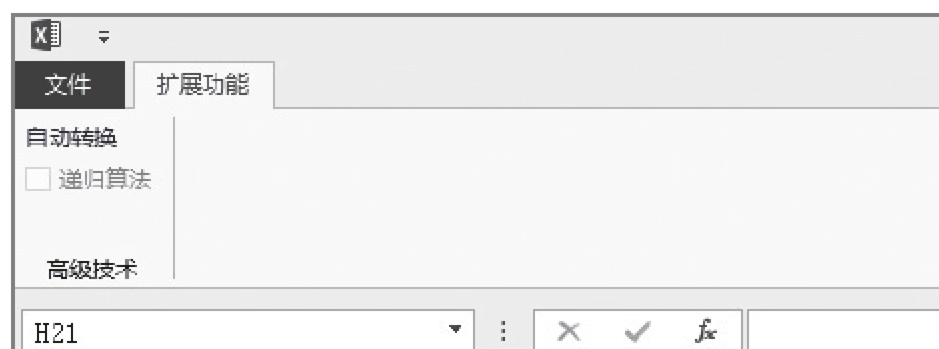


图 5-51　VBA 中变量的值决定控件是否可用

5.4.4　visible-getVisible

visible 是在 XML 中直接指定元素显示或隐藏，而 getVisible 则依赖于 VBA 回调函数。

“实例文档 17.xlsm”的 XML 代码中，group、button、checkBox 都用 getVisible 来设置元素的可见性。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能">
                <group id="Group1" label="高级技术" getVisible="Group_Visible">
                    <button id="Button1" label="自动转换" getVisible="Button_Visible"/>
                    <checkBox id="Check1" label="递归算法" getVisible="Check_Visible"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

相应的 VBA 回调函数如下。

```
Public Sub Group_Visible(control As Office.IRibbonControl, ByRef visible)
    visible = True
End Sub
Public Sub Button_Visible(control As Office.IRibbonControl, ByRef visible)
    visible = False
End Sub
Public Sub Check_Visible(control As Office.IRibbonControl, ByRef visible)
    visible = True
End Sub
```

代码分析：回调函数中的 ByRef visible 参数用于决定控件是否可见。本例把 Button 的

visible 设置为 False，因此在 Excel 中看不到该按钮，如图 5-52 所示。

visible-getVisible 除了可以显示 / 隐藏 group 下面的控件外，还可以隐藏内置选项卡、组、控件等。

例如下面的 XML 首先隐藏“开始”选项卡中的“字体”组，然后隐藏“公式”选项卡。

图 5-52　getVisible 属性的应用

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab idMso="TabHome">
                <group idMso="GroupFont" visible="false">
                </group>
            </tab>
            <tab idMso="TabFormulas" visible="false">
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-53 所示。

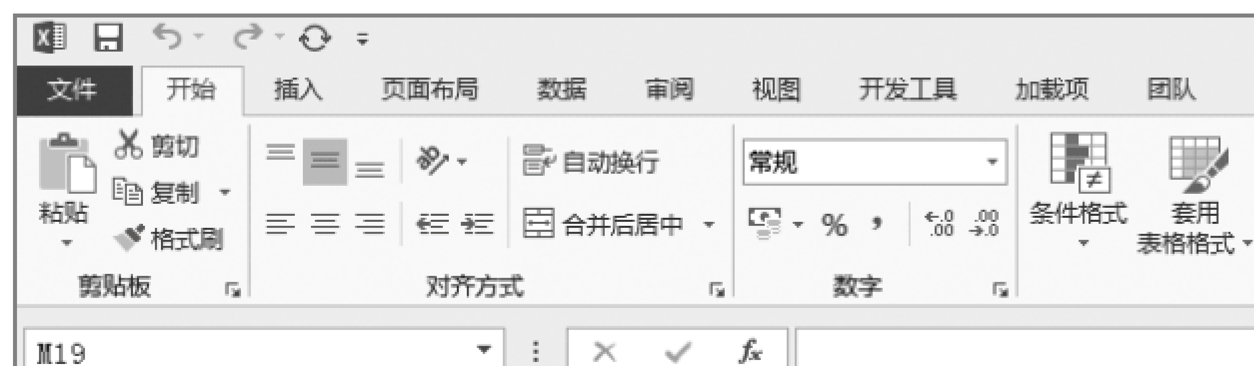


图 5-53　使用 visible 属性决定可见性

注意　并非所有内置元素都可以修改其各种属性。

5.4.5　label-getLabel

label 是元素的静态属性，用于多种元素，例如 tab、group、button 等可以设置 label 属性，用来设置控件的标题文字。

与 label 属性对应的回调属性是 getLabel。

“实例文档 18.xlsm”中的 XML 如下所示。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" getLabel="Tab_Label">
                <group id="Group1" getLabel="Group_Label">
                    <labelControl id="Label1" getLabel="Label_Label"/>
```

```
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

相应的 VBA 回调函数如下。

```
Public Sub Tab_Label(control As Office.IRibbonControl, ByRef label)
    label = Now
End Sub
Public Sub Group_Label(control As Office.IRibbonControl, ByRef label)
    label = Date
End Sub
Public Sub Label_Label(control As Office.IRibbonControl, ByRef label)
    label = Time
End Sub
```

代码分析：XML 中包含 tab、group、labelControl 三个元素，每个元素的标题属性都是动态的，取决于 VBA 中的回调函数。

各个回调函数中的参数 control 是指控件本身，而 label 是指返回的标题。

打开该工作簿，可以看到各个元素分别显示了当前的日期和时间，如图 5-54 所示。



图 5-54　getLabel 属性的应用

细心的读者可能发现以上三个回调函数的结构是一样的，不同的是回调函数名不同。其实，如果回调函数的参数个数、各参数类型一样，多个控件可以共享同一个回调函数。

“实例文档 19.xlsm”中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" getLabel="ReturnLabel">
                <group id="Group1" getLabel="ReturnLabel">
                    <labelControl id="Label1" getLabel="ReturnLabel"/>
                    <button id="Button2" getLabel="ReturnLabel"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

可以看到 XML 中的 4 个元素的 getLabel 指向同一个回调函数：ReturnLabel。因此在 VBA 中只须创建一个回调函数即可。

```
Public Sub ReturnLabel(control As Office.IRibbonControl, ByRef label)
    Select Case control.ID
    Case "Tab1": label = " 工作簿 "
    Case "Group1": label = " 工作表 "
    Case "Label1": label = " 单元格 "
    Case Else: label = " 其他……"
    End Select
End Sub
```

不同控件的 id 一定不同，因此借助控件的 id 属性或者 tag 属性的区别，用 If 或 Select 结构区分开即可。

与上述 XML 对应的 Excel 界面如图 5-55 所示。



图 5-55　多个控件共用同一个回调函数

5.4.6　imageMso-image-getImage

button、menu 等元素都支持图标，为控件设置图标有三种方式，其中 image 和 imageMso 是静态属性。getImage 是回调属性，使用 VBA 过程为控件指定图标。

控件的图标可以是 Office 内置图标（imageMso）或者计算机中的图片文件。

imageMso 与前面讲过的 idMso 有一定的联系，例如要在 XML 中引用 Excel 2013 的【插入 / 符号 / 符号】这个 Ω 形状的按钮，如图 5-56 所示。



图 5-56　引用内置控件的图标

使用 OfficeidMsoViewer 软件可以快速看到该控件的 XML，如图 5-57 所示。

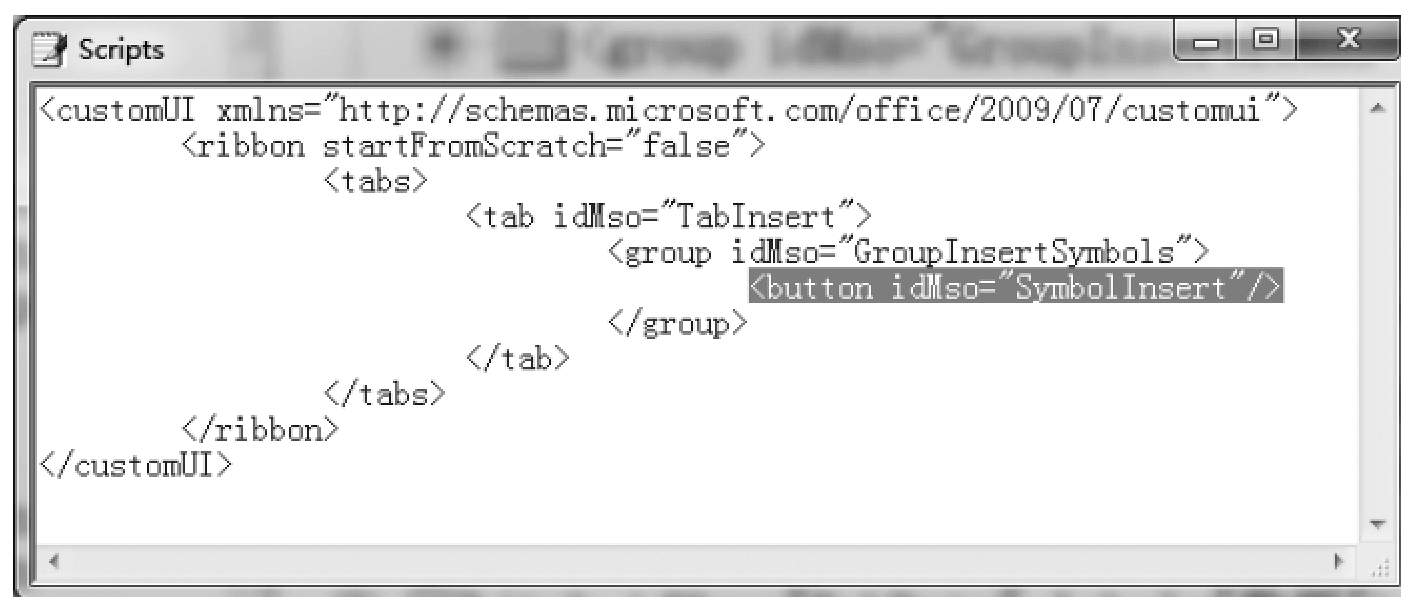


图 5-57　查询内置控件的 idMso

图中可以看到 Ω 的 idMso 是 SymbolInsert，那么以后创建的自定义控件，其 imageMso 属性如果也指定为 SymbolInsert，那么这个新控件的图标也是 Ω。

下面的 XML 代码产生的按钮的图标就是 Ω。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
```

```
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label=" 图标设定 ">
                    <button id="ButtonID3" label=" 我的符号 " imageMso="SymbolInsert"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

以上 XML 代码对应的 Excel 界面如图 5-58 所示。



图 5-58　使用内置控件的 idMso

因此，只需要设定 imageMso 属性，控件的图标就显示为 Office 内置图标。不同版本 Office 的内置图标的名称和数量也不同，Office 2010 和 Office 2013 大约有一万个可以使用的内置图标。

笔者制作的 imageMso7345.xlsm 可以快速查找所有内置图标的名称和外观，如图 5-59 所示。

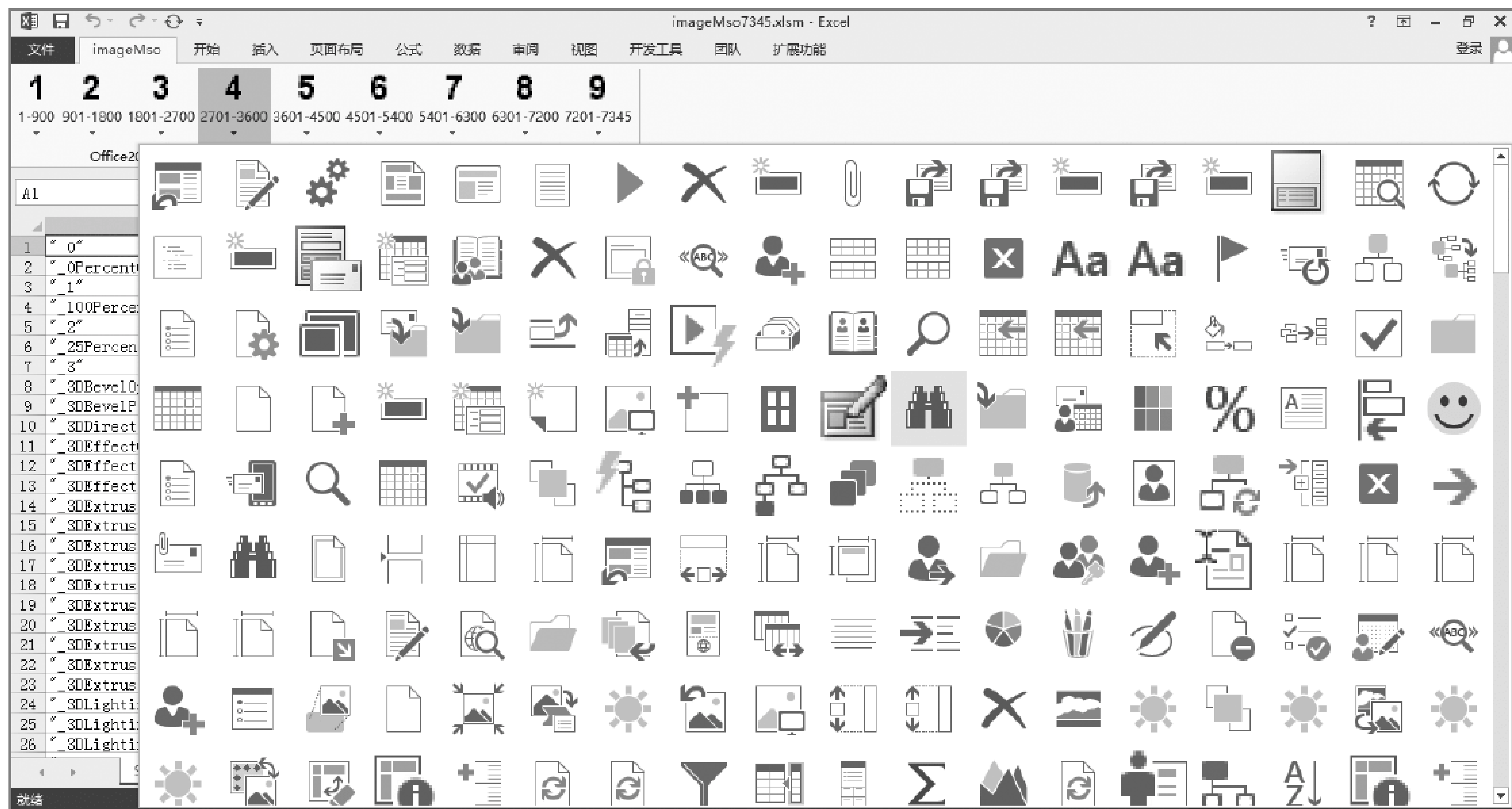


图 5-59　内置图标查看器

在这些内置图标中，有一些是比较简单的图标，例如 26 个大写英文字母 A ～ Z，使用 imageMso="M"，图标就显示为 M。还可以使用 0 ～ 9 这 10 个数字作为图标，但是前面要加下画线。

1. 使用 Office 内置图标

下面的 XML 代码使用字母和数字作为按钮的图标。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label=" 字母图标 ">
                    <button id="Button1" label="x" imageMso="X"/>
                    <button id="Button2" label="y" imageMso="Y"/>
                    <button id="Button3" label="z" imageMso="Z"/>
                </group>
                <group id="Group2" label=" 数字图标 ">
                    <button id="Button4" label="0" imageMso="_0"/>
                    <button id="Button5" label="1" imageMso="_1"/>
                    <button id="Button6" label="2" imageMso="_2"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

以上 XML 代码对应的 Excel 界面如图 5-60 所示。



图 5-60　数字、字母作为图标

2. 使用自定义图标

image 属性规定可以使用自定义图片作为控件的图标，而不是 Office 内置图标。下面讲解用 Custom UI Editor for Microsoft Office 软件向 Office 文档中压入自定义图片并应用于功能区控件中。

在 Excel 2013 中新建一个空白工作簿，另存为“实例文档 20.xlsm”，然后关闭该工作簿文件。

启动 Custom UI Editor 软件，单击菜单【 File/Open 】，打开“实例文档 20.xlsm”。为了编写 XML 代码，需要再单击菜单【 Insert/Office 2010 Custom UI Part 】，在右侧出现的编辑区写入如下 XML 代码。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label=" 棋子 ">
                    <button id="Button1" label=" 仕 " size="large" image="shi"/>
                    <button id="Button2" label=" 帅 " size="large" image="shuai"/>
                    <button id="Button3" label=" 仕 " size="large" image="shi"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

编写过程如图 5-61 所示。

注意代码中 3 个 Button 的 image 属性，因此还需要向该文档中添加相应名称的 png 格

式图片。

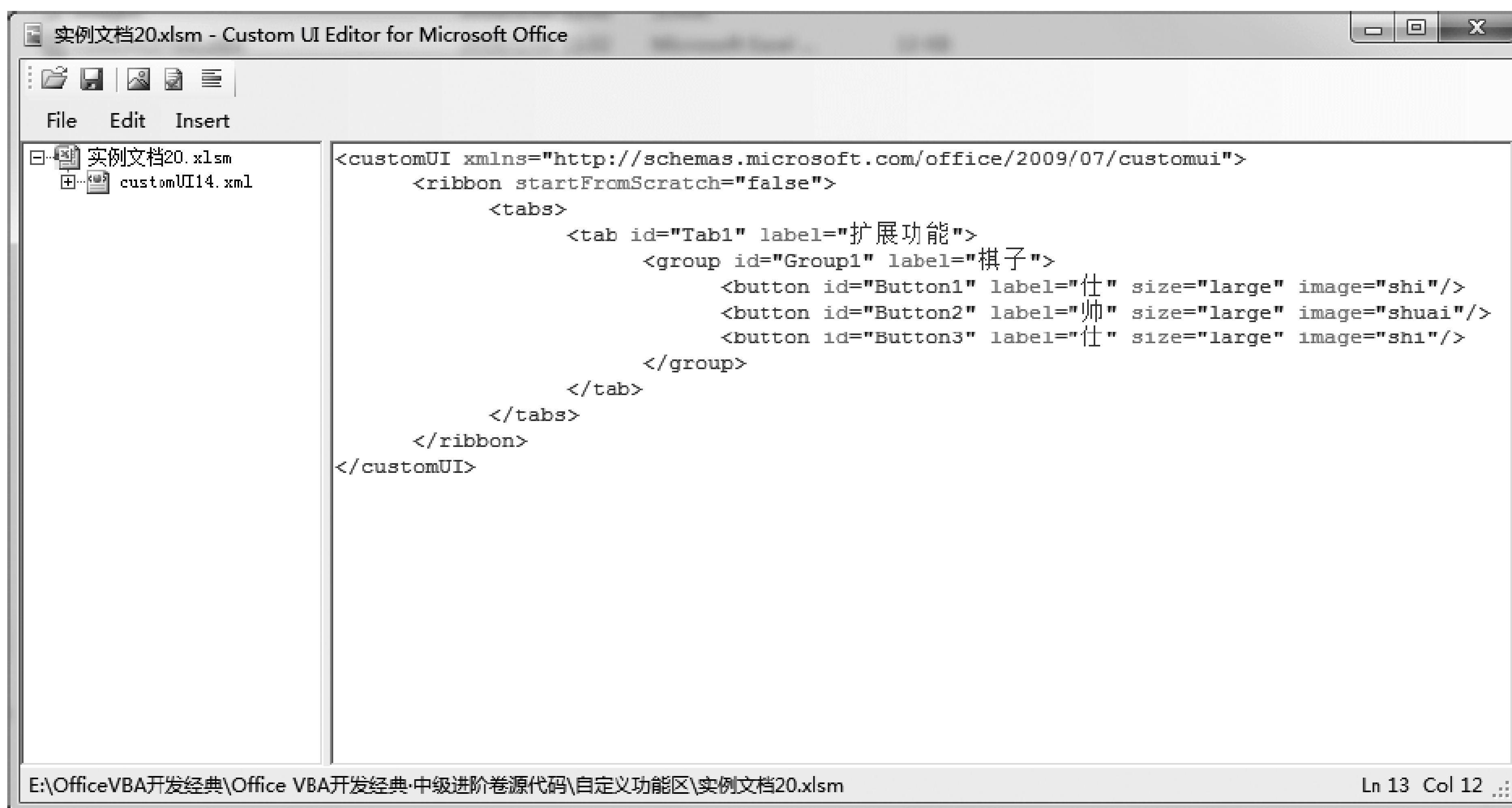


图 5-61　编写 XML 代码

再次单击软件的菜单【 Insert/Icons 】，浏览并选择计算机中的两个 .png 图片：shi.png 和 shuai.png。（.png 图片可以从网上下载，或者把其他图片用图片工具转换为 .png 格式也行。）

此时通过左侧树状结构可以看到加入了两个自定义图片，图片的 id 分别为 shi 和 shuai，如图 5-62 所示。

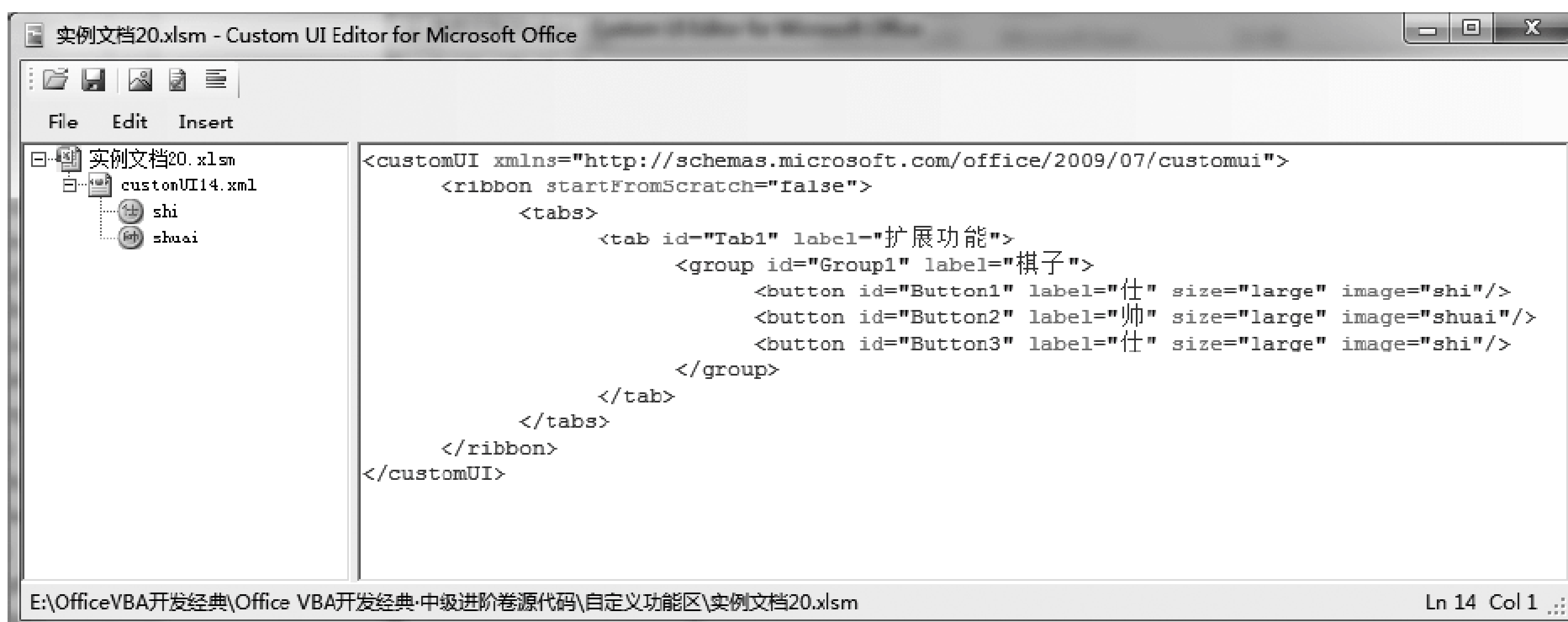


图 5-62　加入自定义图片

编写好 XML 代码并添加了图片后，就可以单击【 File/Save 】保存文档，然后单击【 File/Close 】从软件中关闭工作簿（如果不关闭，在 Excel 中没法打开）。

然后在 Excel 中打开“实例文档 20.xlsm”，如图 5-63 所示。



图 5-63　加入了自定义图标的效果

如果后期需要对 XML、图标进行变更，可以继续使用 Custom UI Editor 这个软件打开文档。

3. 使用 getImage 动态改变图标

getImage 可以通过 VBA 回调函数为控件动态设定图标。图标的来源可以是 Office 内置图标，也可以是计算机中的图片文件。

“实例文档 21.xlsm” 中的 XML 代码如下所示。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label=" 动态图标 ">
                    <button id="Button1" label="Test" size="large" getImage=
"GetJPG"/>
                    <button id="Button2" label="Symbol" size="large" getImage=
"UseBuiltin"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

该 XML 中的两个按钮都用 getImage 属性通过回调函数返回图标，因此 VBA 中的回调函数如下。

```
Public Sub GetJPG(control As Office.IRibbonControl, ByRef image)
    Set image = LoadPicture(ThisWorkbook.Path & "\images\test.jpg")
End Sub
Public Sub UseBuiltin(control As Office.IRibbonControl, ByRef image)
    image = "SymbolInsert"
End Sub
```

代码分析：回调函数中的参数 image 既可以通过 LoadPicture 方法装载本地的图片文件，也可以传入一个 Office 内置图标的名称，其中 SymbolInsert 就是内置图标 Ω。

打开该工作簿，实际效果如图 5-64 所示。



图 5-64　使用 getImage 动态获取图标

前面讲过，gallery 控件通常用于展示图片，因此把 getImage 属性应用于 gallery 控件，就可以很方便地在功能区展示计算机中的图片，也可以制作 imageMso 查看器。

还可以看出，通过 getImage 可以动态地加载计算机中的图片，无须像 image 属性那样把图片压入文档中。

5.4.7　showImage-showLabel

控件的诸多属性中，有一些属性是以单词 show 开头的，这些属性的可取值均为布尔值，而且默认值都是 true。

当 showImage 设置为 false 或 0 时，控件的 image、imageMso 或 getImage 设置的图标均不显示，也就是隐藏图标。

当 showLabel 设置为 false 或 0 时，控件的 Label 或 getLabel 设置的标题文字均不显示，也就是隐藏标题文字。

当控件的 size="large" 时，即使以上两者都设置为 false，也不会隐藏。也就是说，以上两个属性只有当 size="normal" 时有效。

下面的 XML 代码在组中放置了 3 个按钮，其中第 2 个按钮隐藏图标，第 3 个按钮隐藏标题。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label=" 显示和隐藏 ">
                    <button id="Button1" label=" 笑脸 " imageMso="HappyFace"/>
                    <button id="Button2" label=" 笑脸 " imageMso="HappyFace"
showImage="false"/>
                    <button id="Button3" label=" 笑脸 " imageMso="HappyFace"
showLabel="false"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

在 Excel 中可以看到上下并排 3 个按钮，第 2 个按钮没有图标、第 3 个按钮没有标题，如图 5-65 所示。

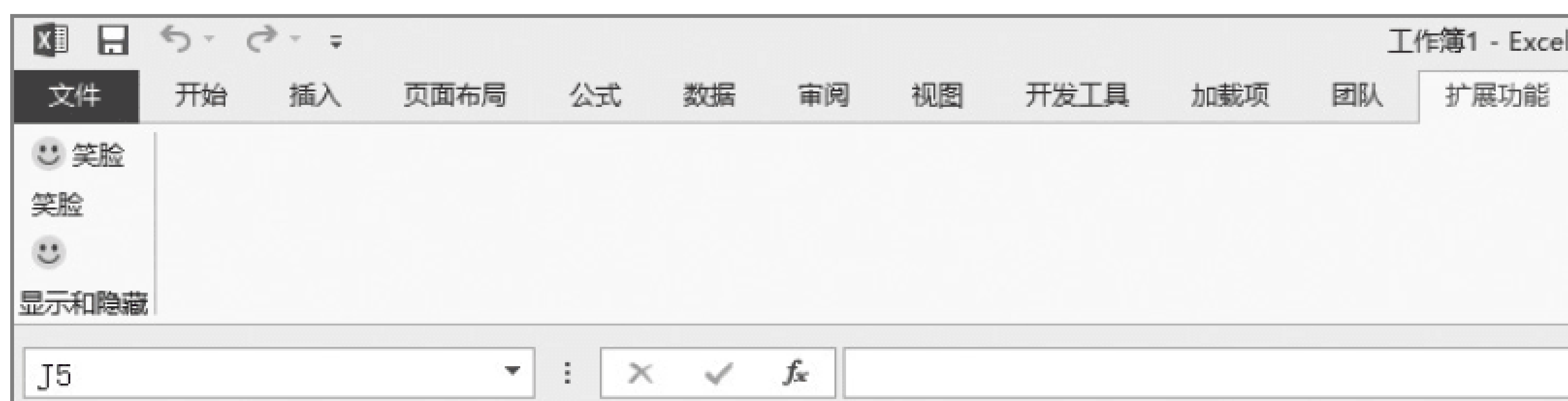


图 5-65　图标和标题的隐藏

5.4.8　onAction

onAction 主要用于规定 button 的回调函数，例如 onAction="Hello", 那么当单击该按钮时，会调用 VBA 中的 Sub Hello 过程，如果不为 button 规定 onAction 属性，则单击该按钮没任何响应，因此一般情况下 onAction 是 button 的一个必备属性。

“实例文档 22.xlsm”中的 XML 如下：

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能">
                <group id="Group1" label="动态图标">
                    <button id="Button1" label="启动 Word" imageMso="MindMap
ExportWord" onAction="LaunchWord"/>
                    <button id="Button2" label="启动 PowerPoint" imageMso=
"MicrosoftPowerPoint" onAction="LaunchPPT"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

VBA 中的回调函数如下。

```
Public Sub LaunchWord(control As Office.IRibbonControl)
    Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub
Public Sub LaunchPPT(control As Office.IRibbonControl)
    Application.ActivateMicrosoftApp xlMicrosoftPowerPoint
End Sub
```

与上述 XML 对应的 Excel 界面如图 5-66 所示。

如果在一个 XML 中存在多个功能相似的按钮，可以把这些按钮的 onAction 设置为同一个 VBA 过程，VBA 过程中根据控件的 id 或 tag 属性的不同，从而执行不同的代码。

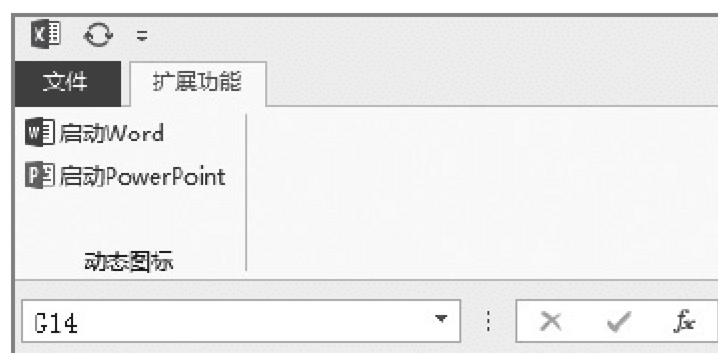


图 5-66　按钮的回调函数

5.4.9　onChange-getText

onChange 和 getText 属性通常用于文本框（editBox）控件。

当向文本框中输入内容，引起文本框内容变化时，会触发 onChange 指定的 VBA 过程。

默认情况下，功能区加载时，各个文本框里都是空白的，但是可以使用 getText 使得文本框能够自动输入内容。

“实例文档 23.xlsm”中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能">
                <group id="Group1" label="文本框">
                    <editBox id="Edit1" label="姓名：" maxLength="6" onChange=
"ContentChanged" getText="UpdateText"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

其中 maxLength 属性规定了文本框最多输入 6 个字符。

VBA 中相应的回调函数如下。

```
Public Sub ContentChanged(control As Office.IRibbonControl, text As String)
    ActiveCell.Value = UCase(text)
End Sub
Public Sub UpdateText(control As Office.IRibbonControl, ByRef text)
    text = "Excel"
End Sub
```

代码分析：ContentChanged 函数的作用是，当用户修改文本框中的内容并按下回车键时，活动单元格会变为文本框内容的大写形式。

UpdateText 函数的作用是，工作簿一打开，文本框里面的内容为“Excel”。

打开该工作簿，效果如图 5-67 所示。

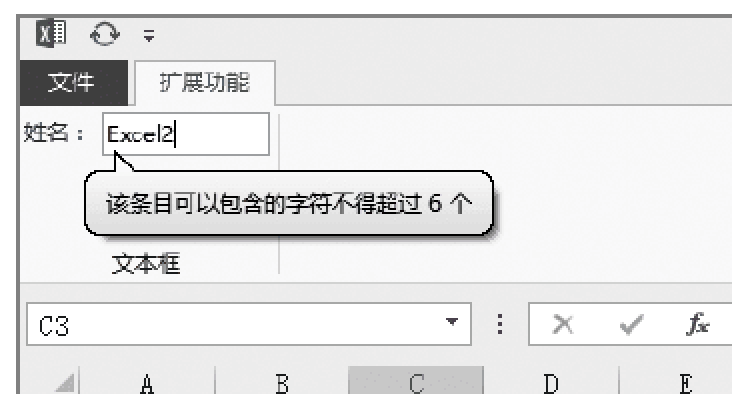


图 5-67　限制输入内容的长度

5.4.10　onLoad

onLoad 只能用于根元素 customUI，其作用是一加载 customUI 就触发 VBA 中 onLoad 对应的回调函数，如果是存储于工作簿中的 customUI，则打开工作簿时触发回调函数，这一点特别类似于 Excel VBA 中的 Workbook_Open 事件。

但是 onLoad 更大的作用是把加载的 customUI 赋给一个 Office.IRibbonUI 类型的公有变量，之后可以方便地刷新 customUI 中控件的状态和属性等。

“实例文档 25.xlsm”中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="customUI_Load">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="customTabA" label="新选项卡 A">
            </tab>
            <tab id="customTabB" label="新选项卡 B">
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

上述 XML 的功能是创建两个自定义选项卡，需要注意的是，该 XML 中包含 onLoad 回调，因此在 VBA 中创建如下回调函数。

```
Public Sub customUI_Load(ribbon As Office.IRibbonUI)
    ribbon.ActivateTab ControlID:="customTabB"
End Sub
```

代码分析：括号中的参数 ribbon As Office.IRibbonUI 就是指整个 customUI。

因此，当一打开工作簿，就自动运行 VBA 中的 customUI_Load 函数，ribbon.ActivateTab

ControlID:="customTabB" 这句表示激活 id 为 customTabB 的自定义选项卡，如图 5-68 所示。



图 5-68　加载 customUI 时自动激活指定的选项卡

5.4.11　IRibbonUI 对象

IRibbonUI 是 Office 对象库下面的成员，该对象只能通过 customUI 的 onLoad 对应的回调函数返回。使用 IRibbonUI 对象的若干方法，可以在 customUI 加载之后反复、多次更新控件的属性。

IRibbonUI 对象的常用方法如下。

- ActivateTab：激活自定义选项卡。
- ActivateTabMso：激活内置选项卡。
- Invalidate：刷新 customUI 的所有元素（重新运行所有回调函数）。
- InvalidateControl：只刷新指定 ControlID 的元素。

为了让 onLoad 回调函数返回的 ribbon 能够被多次访问，通常把 ribbon 赋给模块中的 public 类型变量。

“实例文档 26.xlsm” 中的 XML 如下。

```
    <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="customUI_Load">
        <ribbon startFromScratch="false">
            <tabs>
                <tab id="customTabA" label="新选项卡A">
                </tab>
                <tab id="customTabB" label="新选项卡B">
                    <group id="customGroup1" getLabel="Get_Label">
                             <button id="customButton1" label="激活新选项卡A" size=
"large" onAction="ActivateTabA" imageMso="DataRefreshAll"/>
                             <button id="customButton2" label="激活公式选项卡" size=
"large" onAction="ActivateTabFormulas" imageMso="DataRefreshAll"/>
                             <button id="customButton3" label="刷新组标签文字" size=
"large" onAction="RefreshGroupCaption" imageMso="DataRefreshAll"/>
                    </group>
                </tab>
            </tabs>
        </ribbon>
    </customUI>
```

该 XML 定义了两个自定义选项卡，其中第二个自定义选项卡中包含一个组，组下面包含 3 个自定义按钮。需要注意的是，该 XML 中的 customUI 有个 onLoad 回调，group 元素使用了 getLabel 回调。

VBA 中的代码如下。

```
Public R As Office.IRibbonUI
Public Sub customUI_Load(ribbon As Office.IRibbonUI)
    Set R = ribbon
End Sub
Public Sub ActivateTabA(control As Office.IRibbonControl)
    R.ActivateTab "customTabA"
End Sub
Public Sub ActivateTabFormulas(control As Office.IRibbonControl)
    R.ActivateTabMso "TabFormulas"
End Sub
Public Sub RefreshGroupCaption(control As Office.IRibbonControl)
    R.InvalidateControl "customGroup1"
End Sub
Public Sub Get_Label(control As Office.IRibbonControl, ByRef label)
    label = Time
End Sub
```

代码分析：模块中共包含 5 个函数，R 是模块中的公有变量，当功能区一加载就把整个功能区赋给变量 R，其目的是可以让 R 在其他过程中也能发挥作用。

当单击第 1 个按钮时，激活自定义选项卡“新选项卡 A”；当单击第 2 个按钮时，激活“公式”内置选项卡；当单击第 3 个按钮时，刷新 id 为 customGroup1 的组，意味着要重新运行该 group 涉及的所有回调函数。因此，每当单击第 3 个按钮，会看到组的标签显示当前时间，如图 5-69 所示。

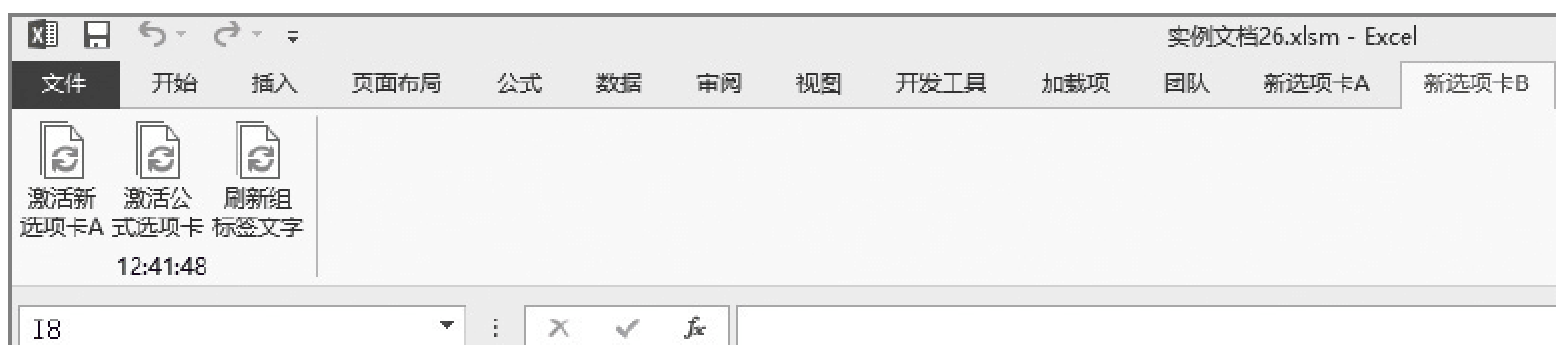


图 5-69　动态刷新功能区

InvalidateControl 方法更新指定 id 的控件，Invalidate 方法则会重新运行 customUI 中的所有回调函数。因此，经常利用这个技术，实现通过 VBA 更改 customUI 控件的属性和状态。

“实例文档 27.xlsm”中的 XML 如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="customUI_Load">
        <ribbon startFromScratch="false">
            <tabs>
                <tab id="customTabA" label="新选项卡A" getVisible="Tab_Visible">
                </tab>
                <tab id="customTabB" label="新选项卡B">
                    <group id="customGroup1" label="动态组">
                        <button id="customButton1" label="按钮" getEnabled="Button_
Enabled"/>
                        <checkBox id="customCheck1" label="复选框" getPressed= "Check_
Pressed"/>
                        <labelControl id="customLabel" getLabel="Label_Label"/>
```

```
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

该 XML 中涉及的重要回调有：新选项卡 A 的 getVisible 来控制该选项卡是否显示，按钮的 getEnabled 属性来控制按钮是否可用，复选框的 getPressed 属性控制该复选框是否勾选，标签控件的 getLabel 控制该标签显示的内容。

对应的 VBA 回调如下。

```
Public R As Office.IRibbonUI
Public Sub customUI_Load(ribbon As Office.IRibbonUI)
    Set R = ribbon
    R.ActivateTab "customTabB"
End Sub
Public Sub Tab_Visible(control As Office.IRibbonControl, ByRef visible)
    visible = Sheet1.Range("B1").Value
End Sub
Public Sub Button_Enabled(control As Office.IRibbonControl, ByRef enabled)
    enabled = Sheet1.Range("B2").Value
End Sub
Public Sub Check_Pressed(control As Office.IRibbonControl, ByRef returnValue)
    returnValue = Sheet1.Range("B3").Value
End Sub
Public Sub Label_Label(control As Office.IRibbonControl, ByRef label)
    label = Sheet1.Range("B4").Value
End Sub
```

代码分析：可以看出，以上各个回调函数中都把单元格的内容传递回去，也就是控件的各种属性取决于单元格的值。

工作表上的“更新状态”按钮对应的 VBA 过程如下。

```
'工作表按钮
Sub RefreshAll()
    R.Invalidate
End Sub
```

打开该工作簿后，更改单元格 B1:B4 的内容，然后单击“更新状态”按钮，可以看到功能区会随之变化，如图 5-70 所示。

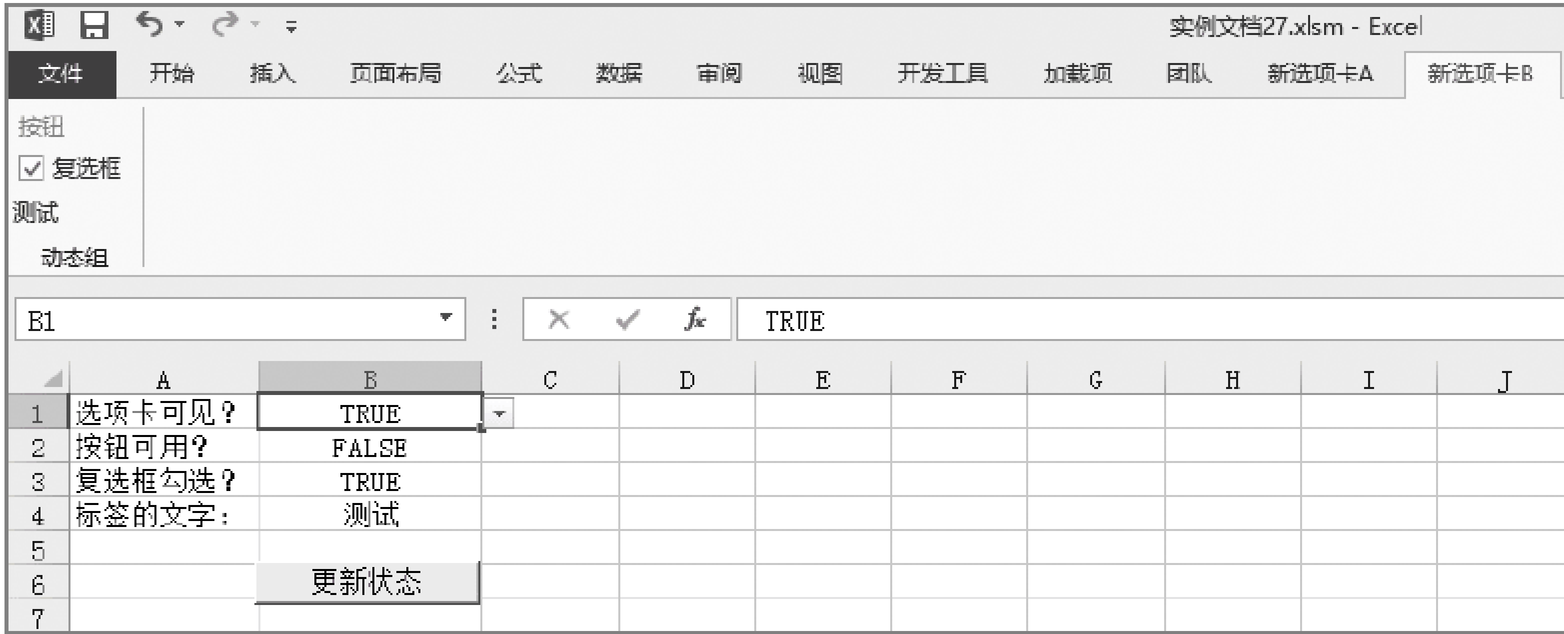


图 5-70　单元格的数值决定功能区的显示状态

5.4.12 screentip-supertip-keytip

screentip、supertip 用于设置控件的提示语，即当鼠标指针悬浮在控件上方时弹出的提示信息。

下面的 XML 代码设置了按钮的 screentip、supertip 属性。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能" keytip="A">
                <group id="Group1" label="扩展组" keytip="B">
                    <button id="Button1" label="提交" supertip="注意：一旦提交不可修改！" screentip="务必确认" keytip="C"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

当鼠标指针移动到按钮时，附近出现一个浮动的提示框，如图 5-71 所示。

keytip 属性用来指定快捷键的。在上面的 XML 中，tab、group、button 的 keytip 依次设置为 A、B、C，那就意味着依次按下键盘上的 Alt、A、B、C，可以快速定位到上述各元素。

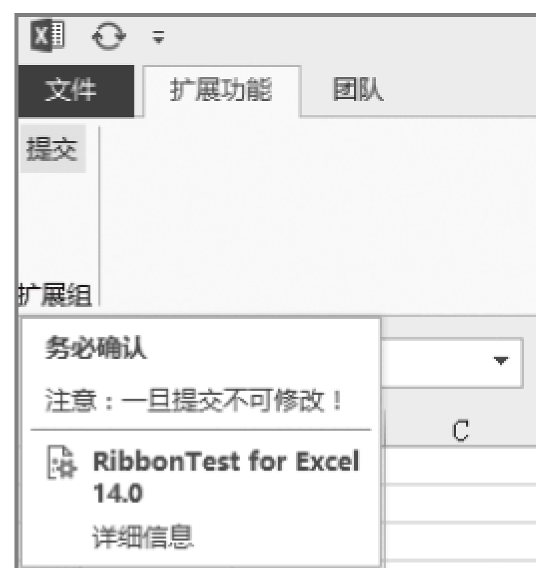


图 5-71　设置提示语

以上 3 个属性对应的动态回调属性分别为 getScreentip、getSupertip、getKeytip。

5.4.13 size

size 属性用来规定控件是否显示为大（large）控件。默认情况下，每列可以垂直放置 3 个控件，如果设置为 large，则每列只能放一个控件。

如果不指定 size 属性，则默认为“normal”，指定为“large”将显示为大控件。

在下面的 XML 代码中，一个 button 是正常尺寸，另一个 button 显示为大控件。

```
<button id="Button1" label="中国" imageMso="SymbolInsert" size="normal"/>
<button id="Button2" label="中华人民&#xA;共和国" imageMso="SymbolInsert" size="large"/>
```

注意　
 是 XML 语言中的换行符，比较长的标题文字就可以分行显示，如图 5-72 所示。

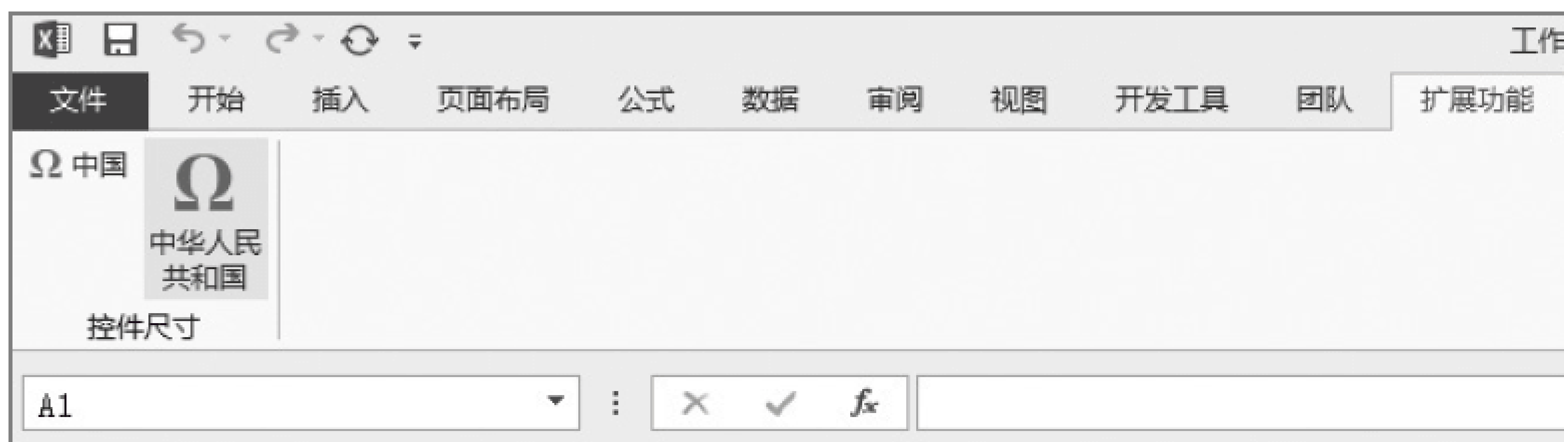


图 5-72　标题在指定位置换行

与 size 属性对应的回调属性为 getSize 属性。

5.4.14　tag

tag 属性并不表现在控件的外观上，通常利用 tag 属性来标识不同的控件，或者让控件存储一些信息，从而让 VBA 使用这些信息。

“实例文档 24.xlsm”中的 XML 包含 3 个自定义按钮。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="Tab1" label=" 扩展功能 ">
                <group id="Group1" label="tag 属性 ">
                    <button id="Button1" label=" 记事本 " tag="notepad.exe" onAction=
"ShellAPP"/>
                    <button id="Button2" label=" 计算器 " tag="calc.exe" onAction=
"ShellApp"/>
                    <button id="Button3" label=" 画图 " tag="mspaint.exe" onAction=
"ShellApp"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

以上 3 个按钮的 onAction 指向同一个 VBA 回调函数，但它们的 tag 不相同。VBA 中的回调函数如下。

```
Public Sub ShellAPP(control As Office.IRibbonControl)
    Shell control.Tag, vbNormalFocus
End Sub
```

代码分析：control.Tag 就是从 XML 中获取 tag 属性的过程，例如单击了“画图”按钮，那么相当于运行了 Shell "mspaint.exe", vbNormalFocus。

打开该工作簿后，单击任一按钮，会自动启动相应的应用程序，如图 5-73 所示。

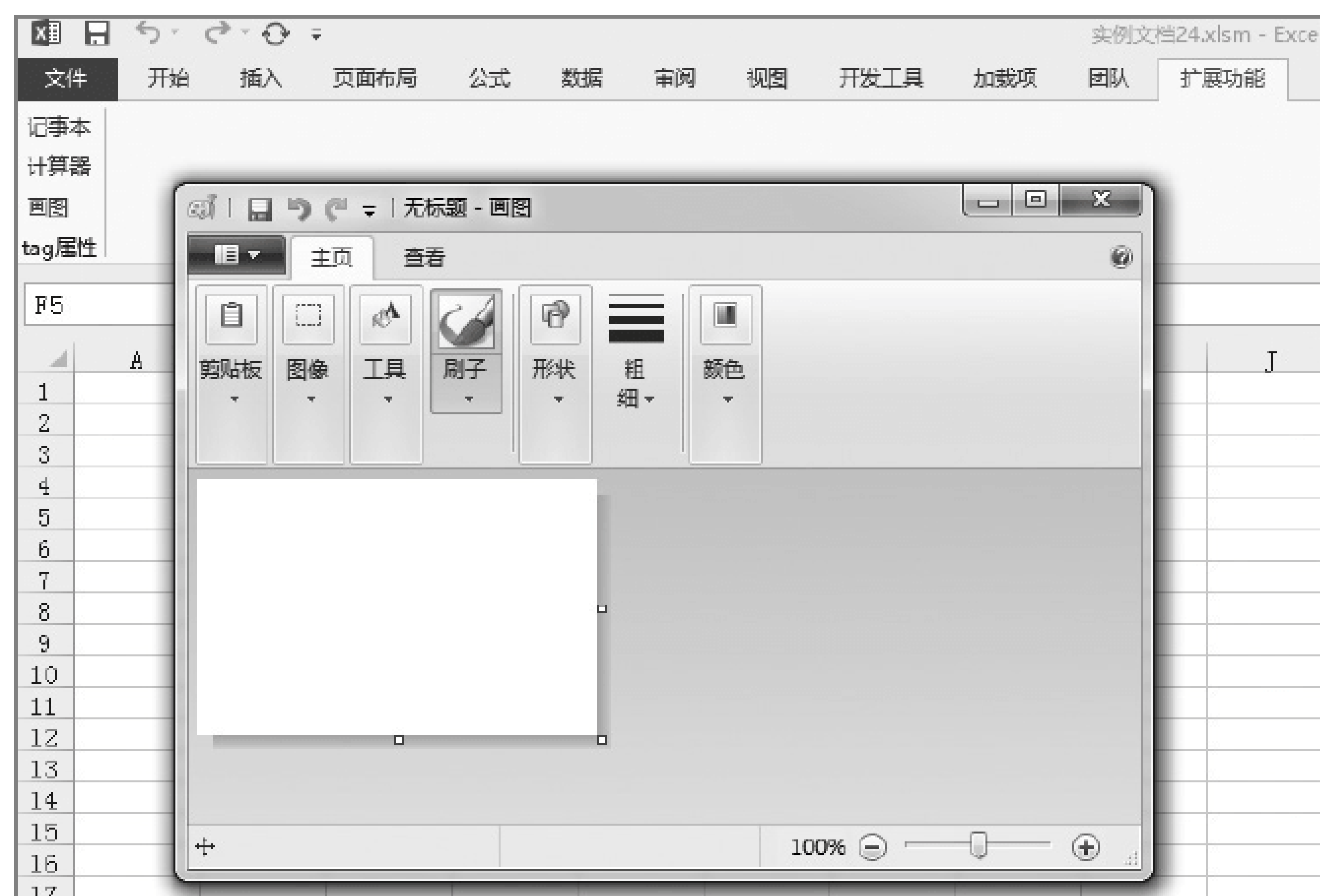


图 5-73　多个控件共用同一个 onAction 回调函数

5.4.15 小结回顾

尽管在 customUI 中允许放置十多种控件，每个控件有诸多属性可以利用，但从开发实用角度讲，只要掌握如下 9 种控件的使用技术就已经足够。

6 个基本控件：labelControl、button、editBox、toggleButton、checkBox 和 dialogBoxLauncher。

3 个复杂控件：comboBox、dropDown、menu。

“实例文档 32.xlsm” 中的 XML 代码充分展示了上述 9 种控件的用法技巧，具体代码如下。

```
    <customUI  xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="customUI_onLoad">
        <ribbon startFromScratch="true">
            <tabs>
                <tab id="Tab1" label="customUI_Demo">
                    <group id="Group1" label=" 简单控件 ">
                        <labelControl id="Label1" label=" 标签控件 "/>
                        <button id="Button1" label=" 按钮 " imageMso="B" onAction=
"Button_OnAction"/>
                        <editBox id="Edit1" label=" 文本框 " getText="EditBox_getText"
onChange="EditBox_onChange"/>
                        <toggleButton id="Toggle1" label=" 切换按钮 " getPressed=
"ToggleButton_getPressed" onAction="toglleButton_onAction"/>
                        <checkBox id="Check1" label=" 复选框 " getPressed="CheckBox_
getPressed" onAction="CheckBox_onAction"/>
                        <dialogBoxLauncher>
                            <button id="Button2" onAction="ShowUserForm"/>
                        </dialogBoxLauncher>
                    </group>
                    <group id="Group2" label=" 复杂控件 ">
                        <comboBox id="Combo1" label=" 组合框 " onChange="ComboBox_
onChange">
                            <item id="comboBox_item1" label=" 子项 1"/>
                            <item id="comboBox_item2" label=" 子项 2"/>
                            <item id="comboBox_item3" label=" 子项 3"/>
                        </comboBox>
                        <dropDown id="dropDown1" label=" 下拉框 " onAction="dropDown_
onAction">
                            <item id="dropDown_item1" label=" 子项 1"/>
                            <item id="dropDown_item2" label=" 子项 2"/>
                            <item id="dropDown_item3" label=" 子项 3"/>
                        </dropDown>
                        <menu id="Menu1" label=" 菜单 " imageMso="M">
                            <button id="menu_button1" label=" 子项 1" onAction="Menu_
Button_onAction"/>
                            <button id="menu_button2" label=" 子项 2" onAction="Menu_
Button_onAction"/>
                            <menuSeparator id="MenuSeparator1" title=" 分隔条 "/>
                            <button id="menu_button4" label=" 子项 4" onAction="Menu_
Button_onAction"/>
                        </menu>
```

```
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

上述 XML 代码对应的 VBA 回调函数如下。

```
'customUI 回调函数模块
Public R As Office.IRibbonUI
Public Sub customUI_onLoad(ribbon As Office.IRibbonUI)
    Set R = ribbon
End Sub
Public Sub Button_OnAction(control As Office.IRibbonControl)
    MsgBox control.ID
End Sub
Public Sub EditBox_getText(control As Office.IRibbonControl, ByRef text)
    text = ActiveSheet.Name
End Sub
Public Sub EditBox_onChange(control As Office.IRibbonControl, text As String)
    MsgBox text
End Sub
Public Sub ToggleButton_getPressed(control As Office.IRibbonControl, ByRef
returnValue)
    returnValue = True
End Sub
Public Sub toglleButton_onAction(control As Office.IRibbonControl, pressed As
Boolean)
    MsgBox pressed
End Sub
Public Sub CheckBox_getPressed(control As Office.IRibbonControl, ByRef
returnValue)
    returnValue = True
End Sub
Public Sub CheckBox_onAction(control As Office.IRibbonControl, pressed As
Boolean)
    MsgBox pressed
End Sub
Public Sub ShowUserForm(control As Office.IRibbonControl)
    UserForm1.Show
End Sub
Public Sub ComboBox_onChange(control As Office.IRibbonControl, text As String)
    MsgBox text
End Sub
Public Sub dropDown_onAction(control As Office.IRibbonControl, selectedId As
String, selectedIndex As Integer)
    MsgBox "所选条目的 ID: " & selectedId & vbNewLine & "所选条目的序号: " & selectedIndex
End Sub
Public Sub Menu_Button_onAction(control As Office.IRibbonControl)
    MsgBox control.ID
End Sub
```

打开该工作簿的效果如图 5-74 所示。

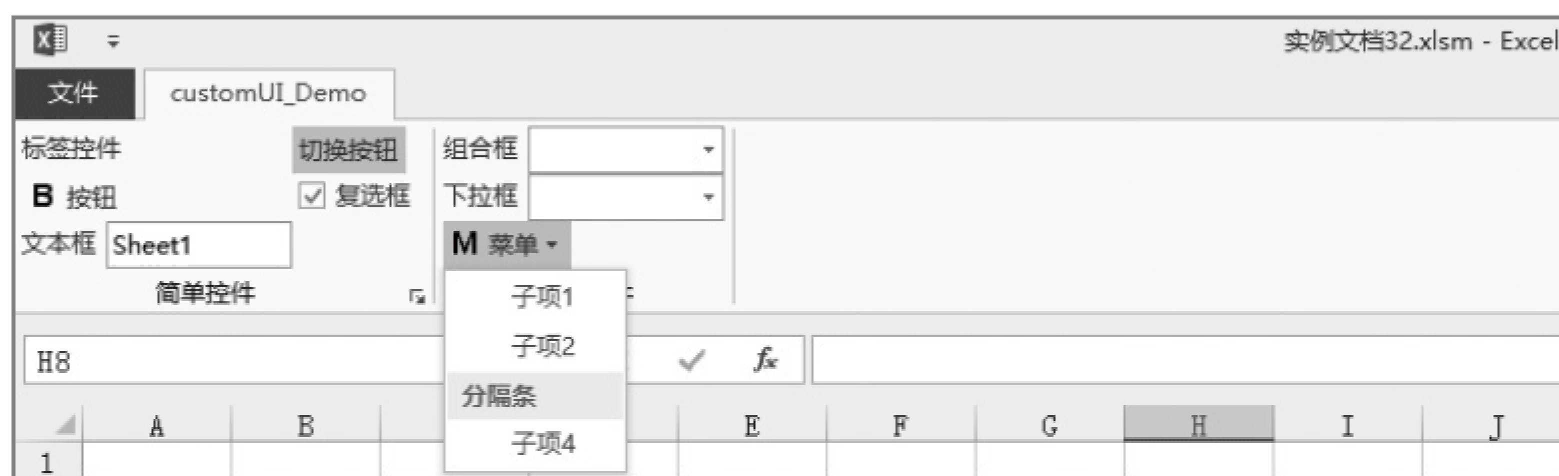


图 5-74　最常用控件的应用展示

5.4.16　customUI 的 XML 代码编写技巧

进行 customUI 的设计开发，能否快速、准确地写出完整、有效的 XML 代码是整个工作的瓶颈。编程人员除了具备 XML 语言的通用知识，还需要了解专门面向 Office customUI 设计的 XML 规范。

以下三个方面是进行 customUI 设计经常遇到而且必须解决的疑难问题。

- ❑ 元素下面允许放置哪些子元素？
- ❑ 元素可以使用哪些属性？
- ❑ 元素的某个属性可以取哪些值？

实际上，不需要死记硬背每个控件的 XML 写法，可以借助带有成员提示的 XML 编辑器（Office Ribbon Editor 或 Visual Studio 的 XML 编辑器）来书写。

例如，想知道 buttonGroup 控件下面允许放置哪些子控件，就可以在其内部输入左尖括号，自动弹出可选菜单，如图 5-75 所示。

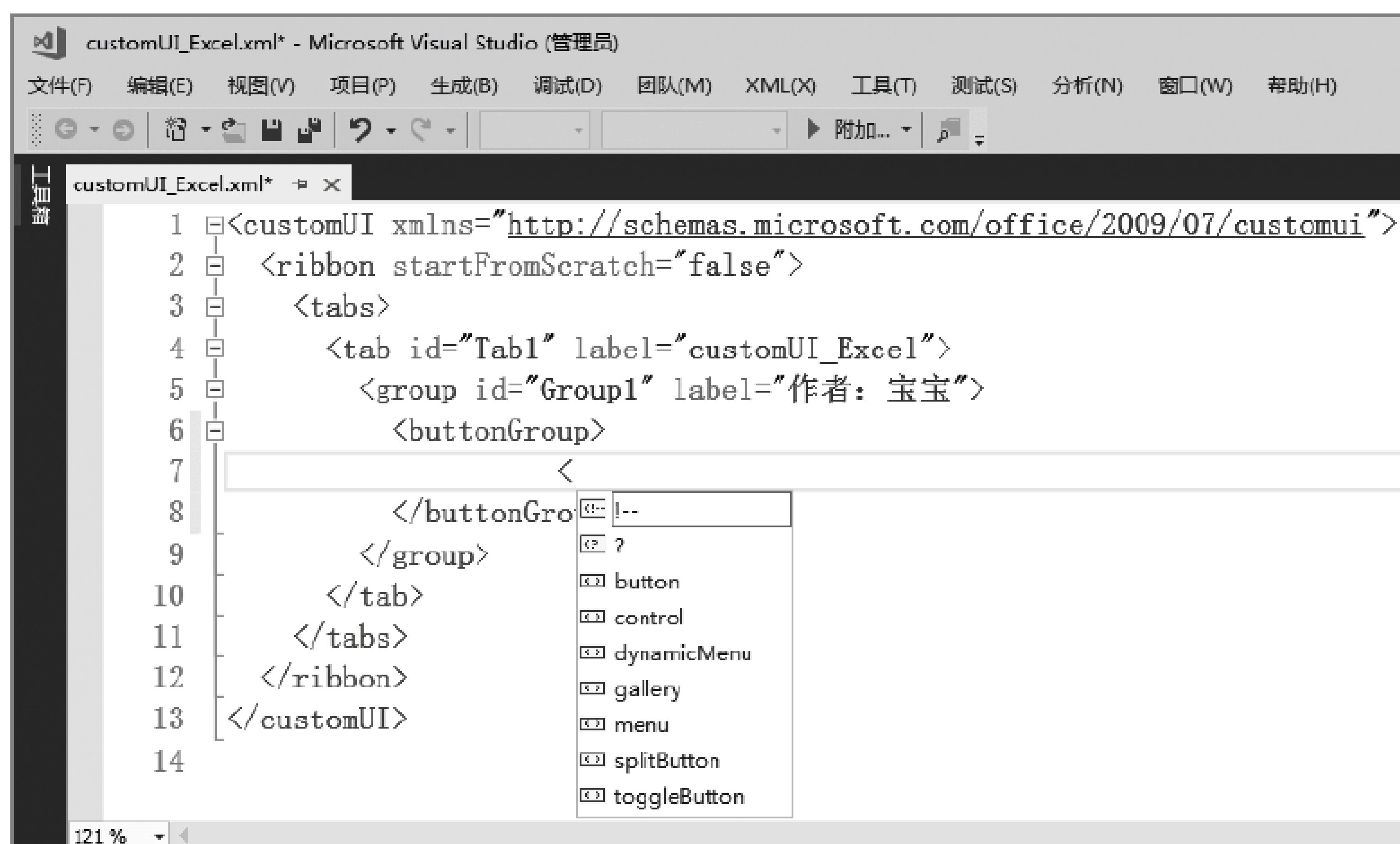


图 5-75　使用 Visual Studio 的 XML 编辑器

不允许添加的控件不会出现在菜单中。

那么，menu 元素可以使用哪些属性呢？

输入 <menu> 后，在单词 menu 后面按一下空格键，就自动列出了所有可用的属性，如图 5-76 所示。

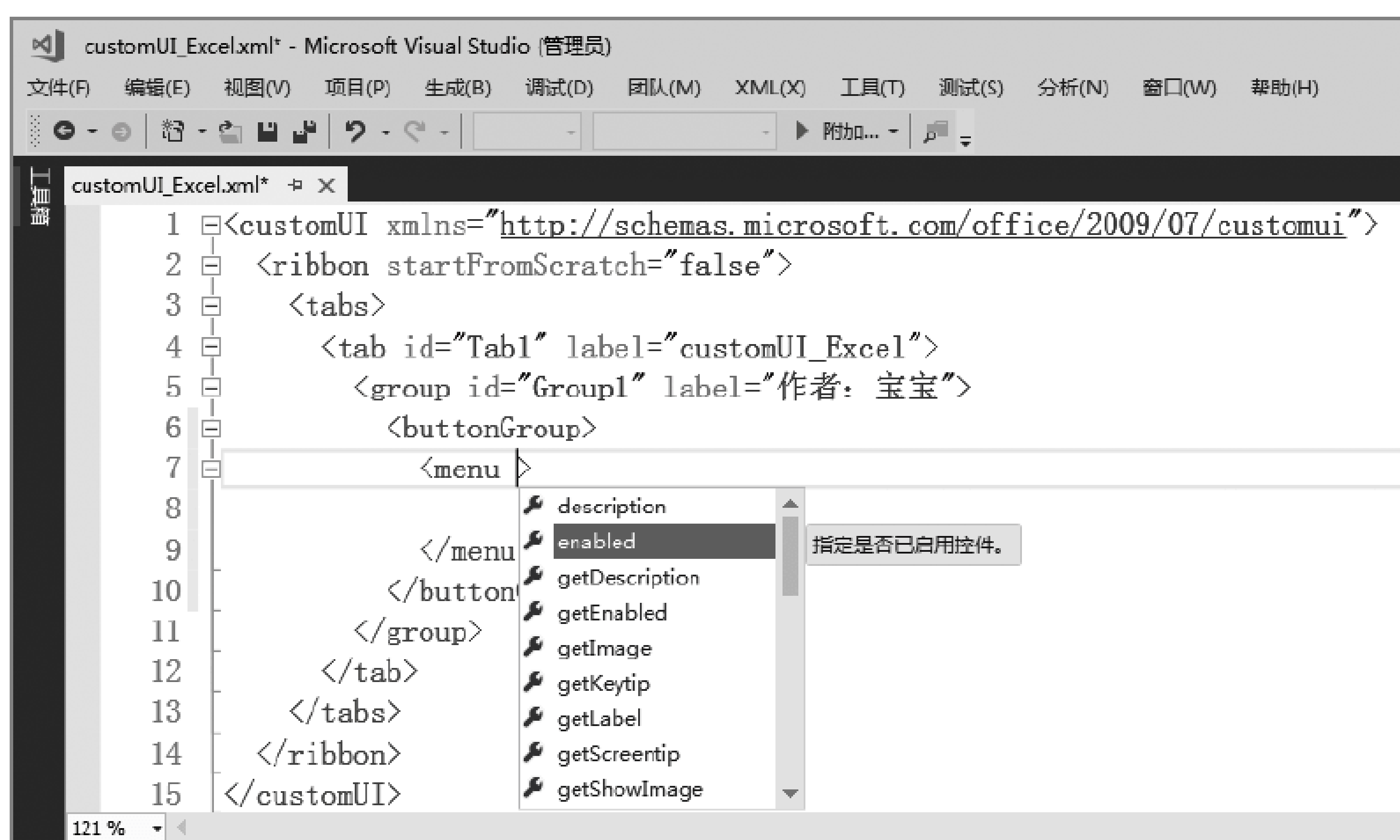


图 5-76　自动弹出可用的属性列表

那么，enabled 属性可以取哪些属性值呢？

输入 <menu enabled="",可以看到只能取菜单中的 4 个值，如图 5-77 所示。

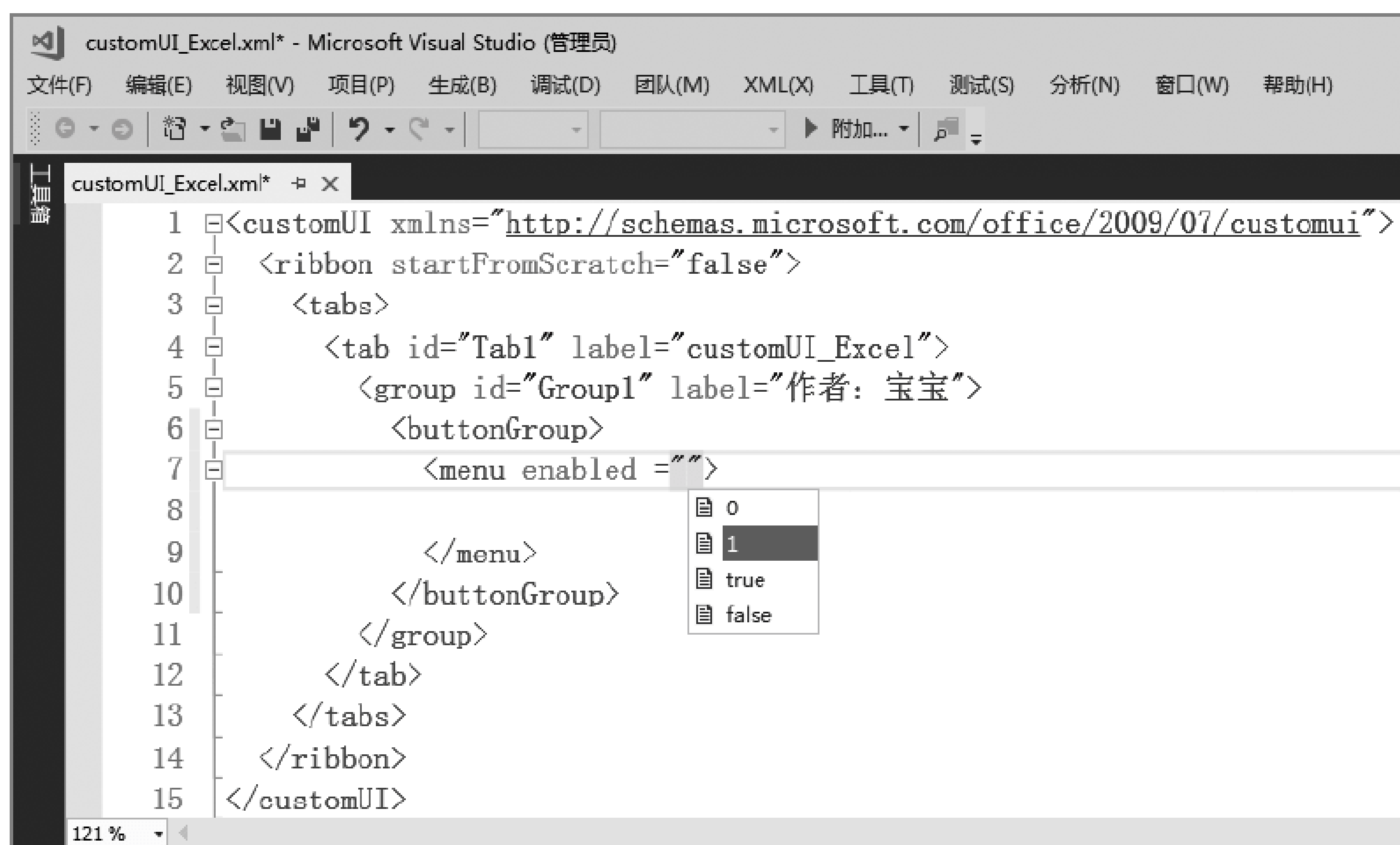


图 5-77　自动弹出属性的可能取值

5.5　使用 Commandbars 对象操作 Office 内置控件

Excel VBA 中的 Application.Commandbars 提供了一些方法，用来执行或读取内置控件。

- ExecuteMso：执行内置控件命令。
- GetEnabledMso：获取内置控件的可用性。
- GetImageMso：获取内置控件的图标。
- GetLabelMso：获取内置控件的标题文字。

❑ GetPressedMso：获取内置控件的按下状态。

❑ GetSceentipMso：获取内置控件的 screentip。

❑ GetSupertipMso：获取内置控件的 supertip。

❑ GetVisibleMso：获取内置控件的可见性。

以上方法的参数都是 idMso 字符串。

5.5.1 获取内置控件属性

这里以中文版 Excel 2013 的“开始”选项卡的“字体”组的“倾斜”按钮为例，通过 OfficeidMsoViewer 软件可以查到其 idMso 为 Italic，如图 5-78 所示。

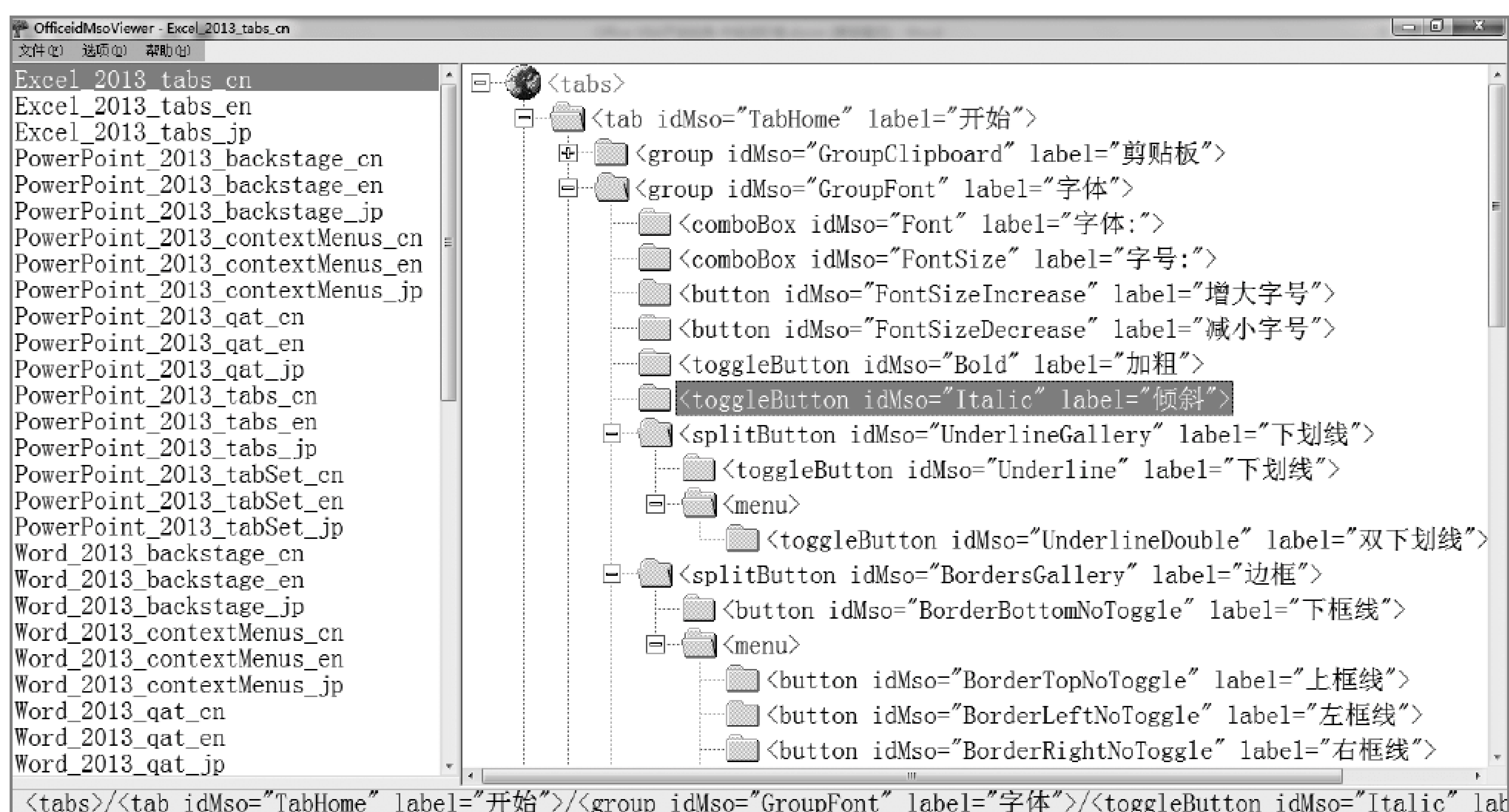


图 5-78 使用 OfficeidMsoViewer 查找内置控件

通过该 idMso 就可以获得该控件目前的状态。

```
Sub GetProperty()
    With Application.CommandBars
        Debug.Print "是否可用：", .GetEnabledMso(idMso:="Italic")
        Debug.Print "Label 是：", .GetLabelMso(idMso:="Italic")
        Debug.Print "是否被按下：", .GetPressedMso(idMso:="Italic")
        Debug.Print "SceenTip 是：", .GetScreentipMso(idMso:="Italic")
        Debug.Print "SuperTip 是：", .GetSupertipMso(idMso:="Italic")
        Debug.Print "是否可见：", .GetVisibleMso(idMso:="Italic")
    End With
End Sub
```

当单击非斜体的单元格，如图 5-79 所示。

运行上述 VBA 过程，立即窗口的运行结果如图 5-80 所示。

如果单击有斜体内容的单元格，再次运行上述过程，结果是不一样的。

理论上讲，使用以上 6 个方法，可以获知 Office 任何一个内置控件的当前属性。

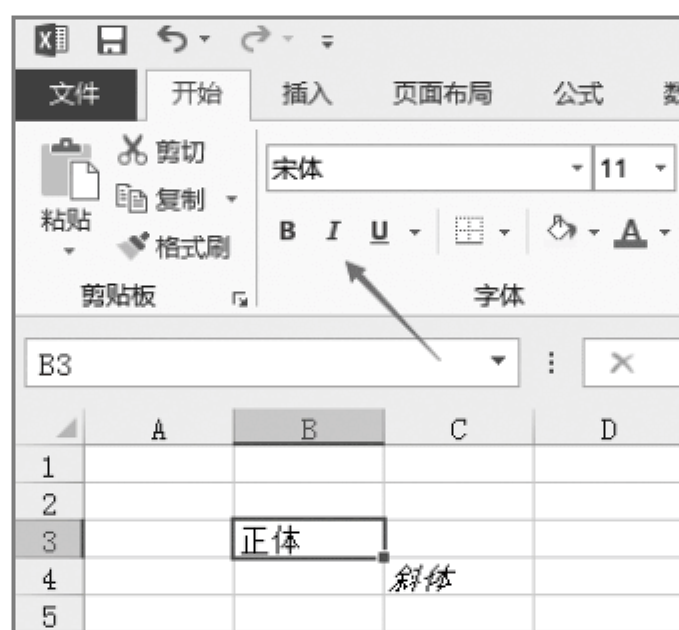


图 5-79　鼠标选中非斜体的单元格

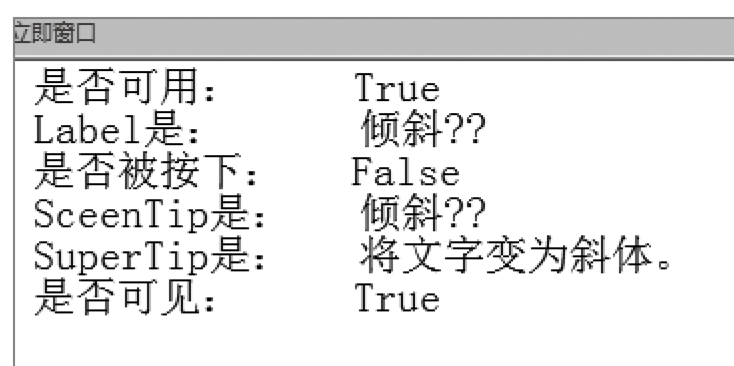


图 5-80　自动获取内置控件的状态属性

5.5.2　自动执行内置控件的命令

如果事先知道某个内置控件的 idMso，则可以使用 ExecuteMso 方法自动执行该控件，而无须单击该控件。

例如，运行下面的代码，自动弹出“页面设置”对话框。

```
Sub AutoExec()
    Application.CommandBars.ExecuteMso ("PageSetupPageDialog")
End Sub
```

5.5.3　获取内置控件的图标

getImageMso 方法会返回一个 IPictureDisp 图形对象，该对象可以赋给 UserForm 的 Image 控件作为图像。

用户窗体上放置一个文本框、一个按钮、一个 image 控件。命令按钮的单击事件如下。

```
Private Sub CommandButton1_Click()
    Dim p As IPictureDisp
    Set p = Application.CommandBars.GetImageMso(idMso: = 
Me.TextBox1.Text, Width:=32, Height:=32)
    Me.Image1.Picture = p
End Sub
```

当在文本框输入任意一个 idMso，单击“显示”按钮，Image 控件就显示为该图标，如图 5-81 所示。



图 5-81　根据内置控件的 idMso 获取图标

以上内容的源代码文件为“实例文档 29.xlsm”。

5.6　自定义快速访问工具栏

自定义快速访问工具栏，需要在 ribbon 元素下面插入 qat 元素，如果 customUI 中包含 qat 部分的定制，ribbon 的 startFromScratch 属性必须设置为 true，也就意味着会隐藏所有内置选项卡。

qat 元素下面是 sharedControls 元素，理论上里面只能包含 button、control 和 separator

三种控件。

下面的 XML 代码在快速访问工具栏中放入内置倾斜按钮，放入一个分隔条，再放置一个自定义按钮。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <qat>
            <sharedControls>
                <control idMso="Italic"/>
                <separator id="Separator1"/>
                <button id="Button1" label="单击" onAction="Click" imageMso=
"HappyFace"/>
            </sharedControls>
        </qat>
    </ribbon>
</customUI>
```

与上述 XML 对应的 Excel 界面如图 5-82 所示。

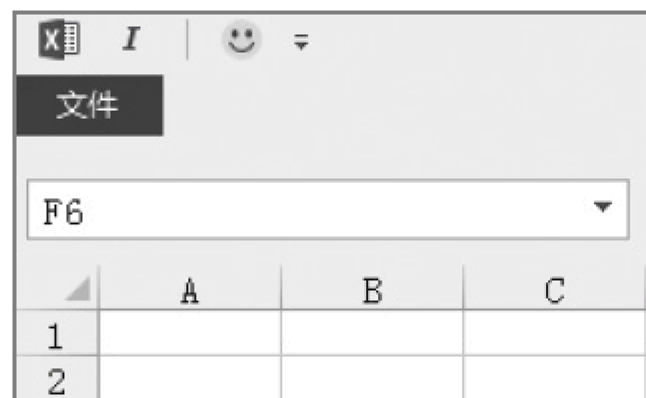


图 5-82 自定义快速访问工具栏

如果要向快速访问工具栏中放入 group 等复杂控件，可以结合自定义常用功能区，重新利用 <tabs> 元素下面的控件。

“实例文档 30.xlsm”包含如下 XML 代码。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label="扩展功能" visible="false">
                <group id="Group1" label="自定义组">
                    <button id="Button1" label="按钮" imageMso="D" onAction="OA"/>
                </group>
            </tab>
        </tabs>
        <qat>
            <sharedControls>
                <control idMso="Bold" imageMso="C"/>
                <control id="Button1"/>
                <control idMso="GroupPageSetup"/>
                <control id="Group1" imageMso="E"/>
            </sharedControls>
        </qat>
    </ribbon>
</customUI>
```

以上 XML 代码首先创建了一个新的自定义选项卡，但是隐藏该选项卡，然后在 <qat> 的部分重新使用前面定义过的 Button1 和 Group1。

VBA 中的回调函数如下。

```
Public Sub OA(control As Office.IRibbonControl)
    ActiveCell.Value = Time
End Sub
```

打开工作簿后，单击最后一个按钮，会弹出一个 group，如图 5-83 所示。

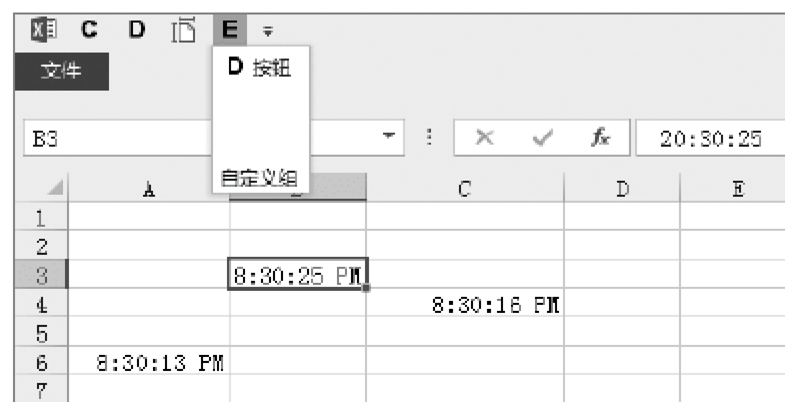


图 5-83 快速访问工具栏中加入 group

对快速访问工具栏进行自定义定制，将强迫隐藏掉内置选项卡，此外，还引起 Excel 的自定义功能区选项对话框无效。这些都给用户带来了很大不便，如图 5-84 所示。



图 5-84　自定义快速访问工具栏造成选项空白

因此，在实际开发过程中，尽量避免对快速访问工具栏进行定制。

5.7　自定义环境功能区

在正常状态下，Office 的环境功能区是隐藏的，只有选择了具体的对象，在常用功能区的右侧显示出特定的功能区。Excel 2013 共有十多个内置的环境功能区，例如 SmartArt 工具、图表工具、绘图工具等，使用 OfficeidMsoViewer 软件可以清晰地看到所有的内置环境功能区及其内部包含的内容，如图 5-85 所示。



图 5-85　查看内置环境功能区

下面以图片（是 Picture，不是 Drawing）的环境功能区为例。例如，在工作表上选中一

个图片，就显示出环境功能区“图片工具”这个 tabSet，可以看到其中包含一个“格式”的 tab，如图 5-86 所示。

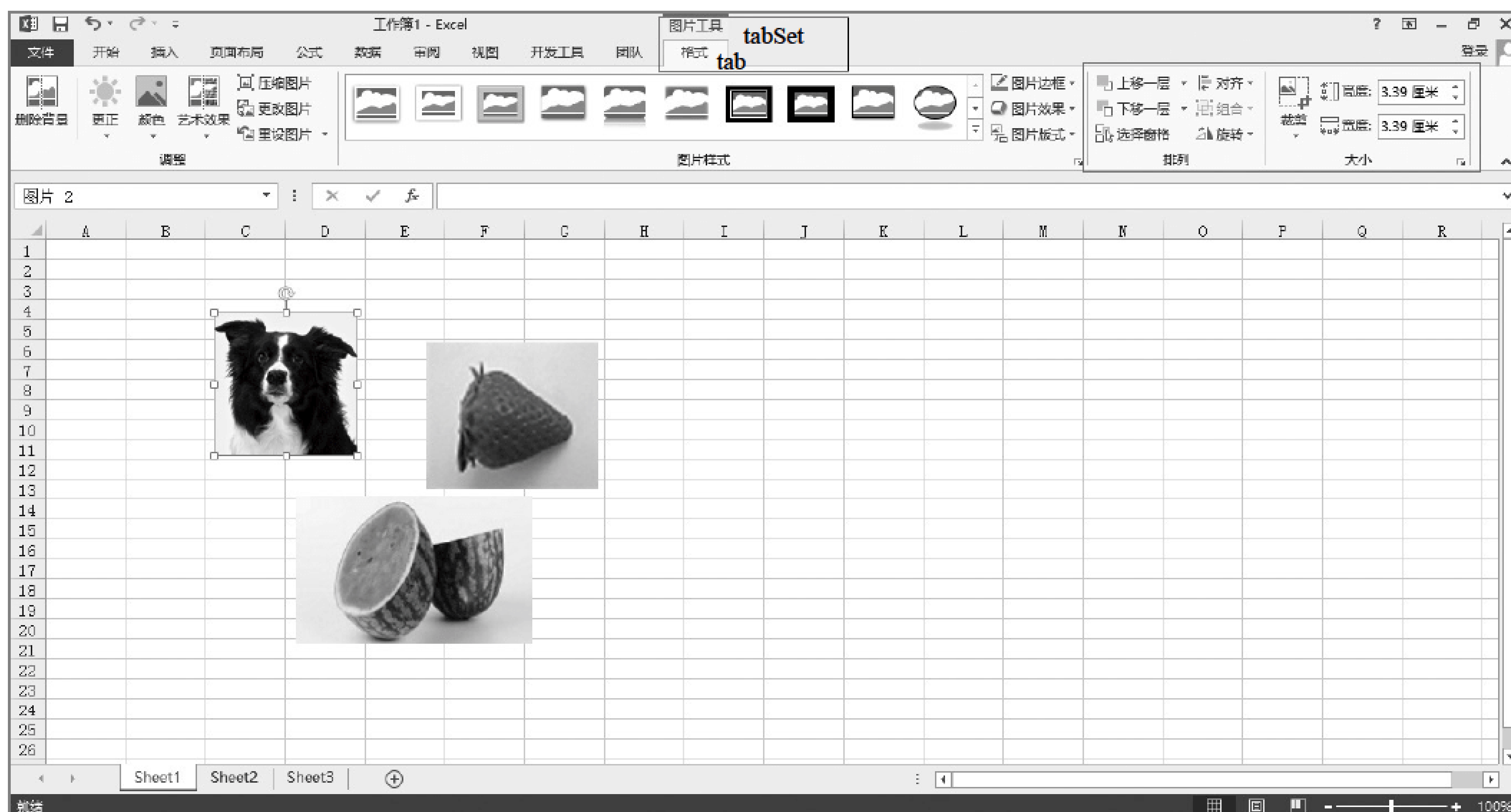


图 5-86　Picture 对象的环境功能区

环境功能区的 XML 结构如图 5-87 所示。

可以看出，从 tabSet 以下，与以前讲过的常用功能区的架构是一样的，也是 tab、group、control 三级结构。tabSet 使用 idMso 属性来指明是哪一个环境功能区，不能创建用户自定义 tabSet，但是可以向内置 tabSet 下面添加自定义 tab，还可以向 tab 中添加自定义 group。

```
- <customUI>
 - <ribbon>
  - <contextualTabs>
   - <tabSet>
    - <tab>
     - <group>
         <control />
       </group>
     </tab>
   </tabSet>
  </contextualTabs>
 </ribbon>
</customUI>
```

图 5-87　环境功能区的 XML 结构

5.7.1　创建自定义选项卡

下面的 XML 代码向图片环境功能区中增加一个自定义 tab。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <contextualTabs>
            <tabSet idMso="TabSetPictureTools">
                <tab id="Tab1" insertBeforeMso="TabPictureToolsFormat" label="
自定义 ">
                </tab>
            </tabSet>
        </contextualTabs>
    </ribbon>
</customUI>
```

代码分析：idMso="TabSetPictureTools" 指的是图片环境功能区，insertBeforeMso= "TabPictureToolsFormat" 表示在“格式”选项卡左侧加入一个自定义选项卡。

Excel 中，选中一个图片后，可以看到多出来一个空白的“自定义”选项卡，如图 5-88 所示。

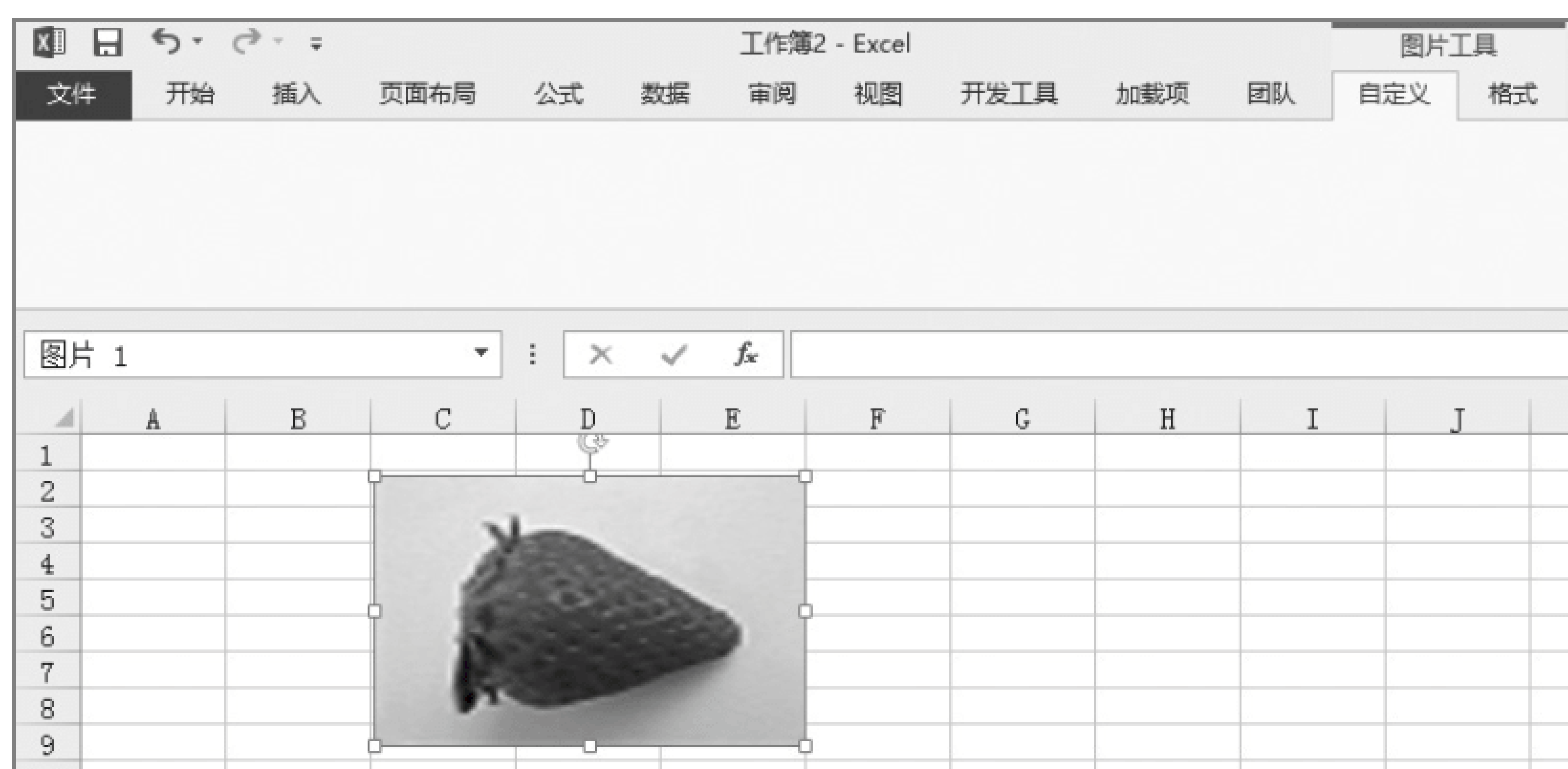


图 5-88　环境功能区中的自定义选项卡

本例只加了一个空白的 tab，读者可以根据需要往里面添加 group 及其 control。

5.7.2　创建自定义组和控件

在图片的环境功能区的“格式”选项卡的右侧可以看到一个名为“大小”的组，里面可以设置所选图片的高度和宽度。

下面往“大小”组左侧插入一个“位置”组，用来设置图片的 Top 和 Left 属性，从而达到精确定位图片的效果。

根据 OfficeidMsoViewer 软件得知，该组的引用方式如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <contextualTabs>
            <tabSet idMso="TabSetPictureTools">
                <tab idMso="TabPictureToolsFormat">
                    <group idMso="GroupPictureSize"/>
                </tab>
            </tabSet>
        </contextualTabs>
    </ribbon>
</customUI>
```

基于上述代码，修改为如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <contextualTabs>
            <tabSet idMso="TabSetPictureTools">
                <tab idMso="TabPictureToolsFormat">
                    <group id="Group1" insertBeforeMso="GroupPictureSize" label=
" 位置 ">
                        <editBox id="Edit1" label="Top" onChange="SetTop"/>
                        <editBox id="Edit2" label="Left" onChange="SetLeft"/>
```

```
                        <button id="Button1" label="对齐到单元格" onAction= "AlignCell"
imageMso="PivotDropAreas"/>
                    </group>
                </tab>
            </tabSet>
        </contextualTabs>
    </ribbon>
</customUI>
```

代码分析：上述 XML 创建了一个“位置”组，添加两个文本框，用来设置所选图片的 Top 和 Left，再添加一个“对齐到单元格”按钮。

上述 XML 代码保存到“实例文档 34.xlsm”中，在其 VBA 模块中写入如下回调。

```
Public sr As Excel.ShapeRange, sp As Excel.Shape
Public Sub SetTop(control As Office.IRibbonControl, text As String)
    Set sr = Application.Selection.ShapeRange
    For Each sp In sr
        sp.Top = CInt(text)
    Next sp
End Sub
Public Sub SetLeft(control As Office.IRibbonControl, text As String)
    Set sr = Application.Selection.ShapeRange
    For Each sp In sr
        sp.Left = CInt(text)
    Next sp
End Sub
Public Sub AlignCell(control As Office.IRibbonControl)
    Set sr = Application.Selection.ShapeRange
    For Each sp In sr
        sp.Left = sp.TopLeftCell.Left
        sp.Top = sp.TopLeftCell.Top
    Next sp
End Sub
```

打开该工作簿，选择若干图片，在 Top 中输入一个数并按下回车键，可以看到所有的图片均顶端对齐到同一位置。如果单击“对齐到单元格”按钮，所有图片都对齐到所属单元格的左上角，如图 5-89 所示。

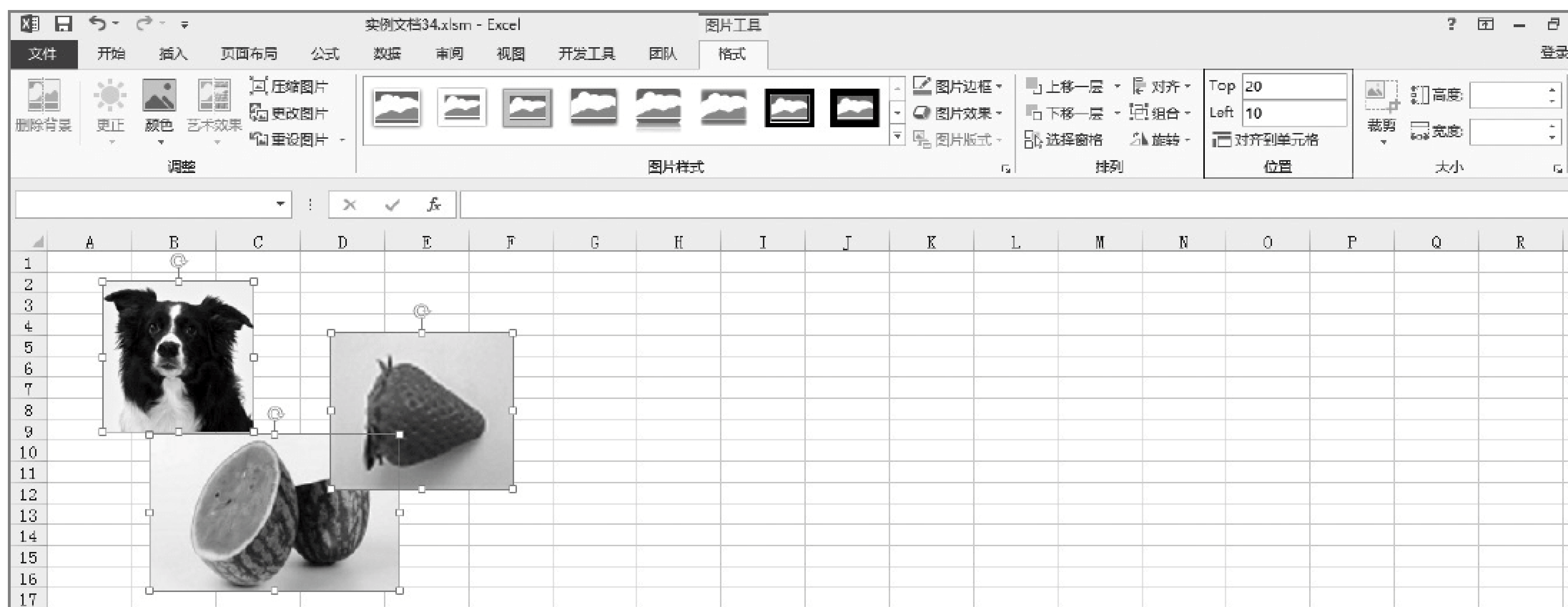


图 5-89　环境功能区中加入自定义组和控件

以上就是自定义环境功能区的技术要点。

5.8　自定义右键菜单

Office 中有很多右键菜单，右键菜单就是指鼠标选中一个对象后单击鼠标右键弹出的菜单。单击的对象不同，弹出的菜单内容也不同。

自定义右键菜单的目的，一是修改右键菜单中的内置控件属性，二是向内置右键菜单中增加用户自定义的部分。

自定义右键菜单的 XML 结构如图 5-90 所示。

```
- <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  - <contextMenus>
    - <contextMenu>
        <control />
      </contextMenu>
    </contextMenus>
  </customUI>
```

图 5-90　右键菜单的 XML 结构

其中，每一个 <contextMenu> 元素都代表一个内置右键菜单。要对内置的右键菜单进行自定义，必须先查询到该菜单的 idMso。

本节以 Excel 的图形右键菜单为例，讲解一下如何对 Office 内置右键菜单进行 customUI 设计。

图形右键菜单是指鼠标在矩形、文本框上单击右键出现的菜单（见图 5-91）。注意：并非图片右键菜单。



图 5-91　Shape 对象的右键菜单

打开 OfficeidMsoViewer 软件，左侧选择 Excel_2013_contextMenus_cn，可以看到该右键菜单的 idMso，以及菜单中包含的各项的 idMso，如图 5-92 所示。

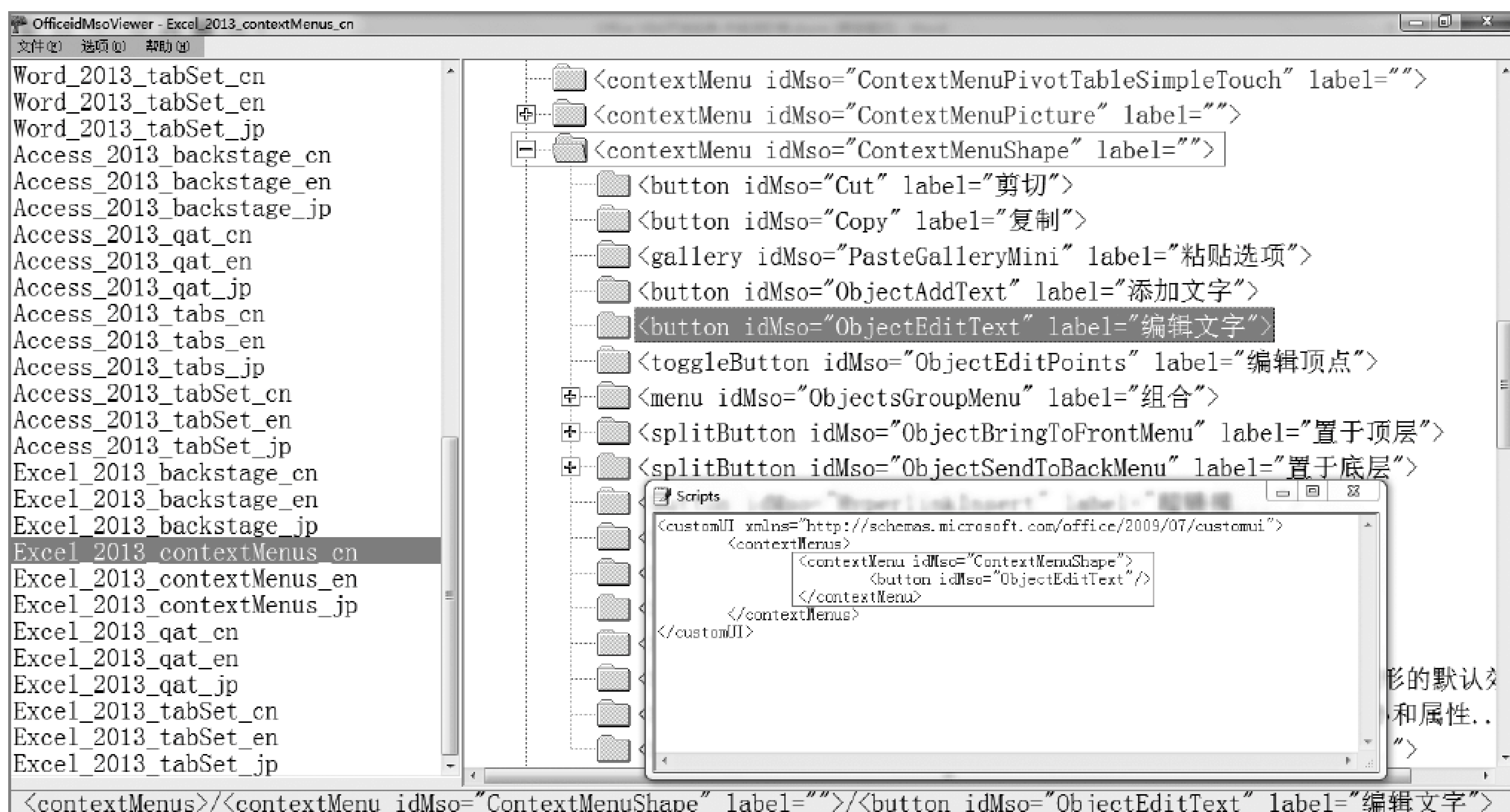


图 5-92　查找内置右键菜单的 XML 定义

可以看到该右键菜单的 idMso 为 ContextMenuShape，其中，“编辑文字”控件是一个 button，它的 idMso 为 ObjectEditText。

5.8.1　修改内置控件状态

知道内置控件 idMso 的前提下，就可以更改其有关属性，下面的 XML 代码把图形的右键菜单里的“剪切”按钮隐藏，并且更改了“编辑文字”按钮的标题文字、可用性和图标。

```
    <customUI  xmlns="http://schemas.microsoft. com/
office/2009/07/customui">
        <contextMenus>
            <contextMenu idMso= "ContextMenuShape">
                <button idMso="Cut" visible= "false"/>
                <button idMso="ObjectEditText" label=
" 文字编辑 " enabled="false" imageMso="HappyFace"/>
            </contextMenu>
        </contextMenus>
    </customUI>
```

图形右键菜单中，“剪切”按钮看不到了，而且“编辑文字”变为灰色，加了一个笑脸图标，如图 5-93 所示。

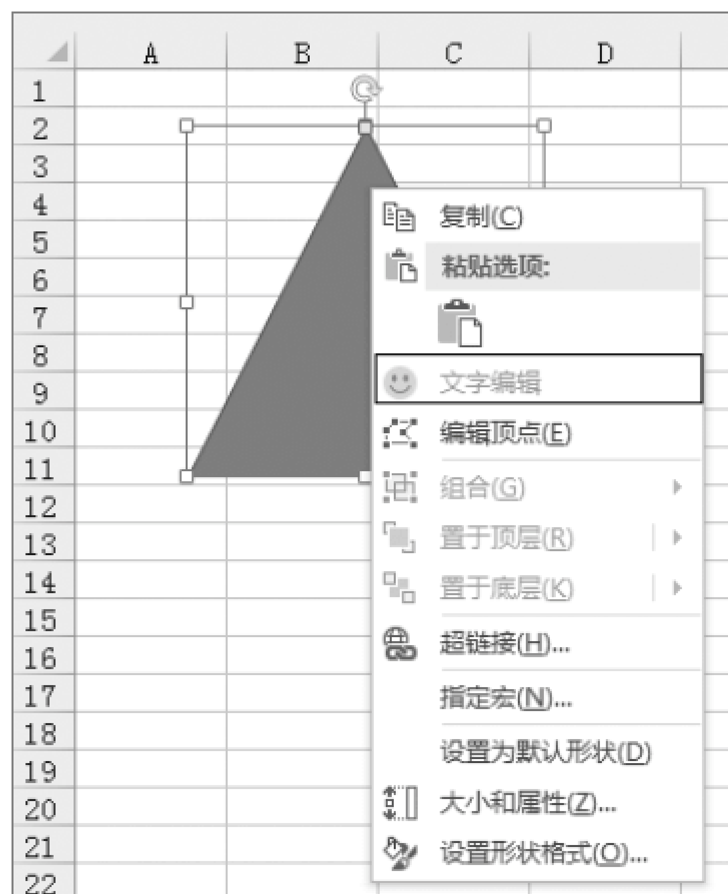


图 5-93　修改右键菜单中内置控件的属性

5.8.2　添加自定义控件

在 ContextMenu 下面允许添加的控件有如下几个。

- ❑ button
- ❑ checkBox
- ❑ dynamicMenu
- ❑ control
- ❑ gallery
- ❑ menu
- ❑ menuSeparator
- ❑ splitButton
- ❑ toggleButton

“实例文档 33.xlsm”包含的 XML 代码向 Excel 图形右键菜单中加入一个 button 和一个 toggleButton。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <contextMenus>
        <contextMenu idMso="ContextMenuShape">
            <button id="Button1" insertAfterMso="Cut" label="隐藏形状" onAction=
"HideShape" imageMso="InsertTag"/>
             <toggleButton id="Toggle1" label="锁定纵横比" onAction="LockAspectRatio"
imageMso="Lock"/>
        </contextMenu>
    </contextMenus>
</customUI>
```

打开该工作簿，在图形上右击，在弹出的菜单中可以看到剪切按钮的下面多了一个“隐藏形状”按钮，同时该菜单的最底端多了一个“锁定纵横比”切换按钮，如图 5-94 所示。



图 5-94　右键菜单中加入自定义控件

这两个控件的回调函数如下。

```
Public Sub HideShape(control As Office.IRibbonControl)
    Dim sp As Shape
    Set sp = Application.Selection.ShapeRange(1)
    sp.Visible = msoFalse
End Sub
Public Sub LockAspectRatio(control As Office.IRibbonControl, pressed As Boolean)
    Dim sp As Shape
    Set sp = Application.Selection.ShapeRange(1)
    sp.LockAspectRatio = Not sp.LockAspectRatio
End Sub
```

5.9 自定义 Office 菜单

在 Office 2010 以上版本中，Office 按钮由“文件”选项卡替代。通过单击“文件”选项卡，可进入 backstage 视图。

与常用功能区相比，backstage 的界面定制更加灵活，元素更加丰富，但是理解难度也相应增大。

开发人员可以对 backstage 视图进行完全扩展，从而允许组织自定义用户界面以满足需求。和开发其他场所一样，用户既可以修改 backstage 视图中的内置控件，也可以从头创建用户自己的元素。

5.9.1 自定义 backstage 视图概述

backstage 视图主要由选项卡（tab）和按钮（button）构成，光从外观上看不出 tab 和 button 的不同之处。例如，Excel 2013 的 backstage 视图中，“保存”“关闭”和“选项”这三个是按钮，而“信息”等是选项卡，如图 5-95 所示。



图 5-95　backstage 视图

单击一个按钮，通常会弹出一个对话框或者执行一个命令，但是单击一个选项卡，会在右侧出现相应的组和控件。

Excel 2013 的 backstage 视图中一部分内置选项卡和按钮信息如表 5-3 所示。

表 5-3　Excel 2013 的 backstage 视图中部分选项卡和按钮信息

idMso	标　题	类　型
TabInfo	信息	tab
TabNew	新建	tab
FileSave	保存	button
TabSave	另存为	tab
TabPrint	打印	tab
TabShare	共享	tab
TabPublish	导出	tab
FileClose	关闭	button
ApplicationOptionsDialog	选项	button

知道内置控件的 idMso 后，就可以更改内置控件的属性，或者添加用户自定义控件。

例如，下面的 XML 代码对 Excel 2013 的 backstage 进行了 4 处自定义。

```
    <customUI xmlns="http://schemas.microsoft.com/
office/2009/07/customui">
        <backstage>
            <button idMso="FileSave" label="Save"
imageMso="FileSave"/>
            <tab idMso="TabShare" label=" 分享 "
insertBeforeMso="TabNew">
            </tab>
            <button id="Button1" label=" 按钮 "/>
            <tab id="Tab1" label=" 选项卡 ">
            </tab>
        </backstage>
    </customUI>
```

修改内置“保存”按钮的标题为“ Save”，修改内置选项卡“共享”为“分享”，加入一个新 button，再加入一个新 tab，并且把新选项卡置于信息选项卡之上，如图 5-96 所示。



图 5-96　自定义 backstage 主菜单

5.9.2　backstage 的 XML 架构

自定义 backstage 的重点和难点是向自定义 tab 中添加元素。与定制常用功能区有所不同，backstage 中的 tab 下面只能是 firstColumn 或 secondColumn，分别表示第 1 列和第 2 列。

firstColumn 下面可以是 group、taskGroup 或 taskFormGroup 三者之一。这三个分别表示

不同的布局风格。

用于自定义 backstage 的典型 XML 结构如图 5-97 所示。

```
- <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  - <backstage>
      <button />
    - <tab>
      - <firstColumn>
        - <group>
            <topItems />
          </group>
        </firstColumn>
      </tab>
    </backstage>
  </customUI>
```

图 5-97　自定义 backstage 的 XML 结构

5.9.3　group 风格

tab 的下一级是 firstColumn 或 secondColumn，firstColumn 的下一级可以是 group、taskGroup 或 taskFormGroup。

group 的下一级可以是 bottomItems、primaryItem、topItems 三者之一。

primaryItem 下面只能放置 button 或 menu 控件，topItems 下面则可以放置很多种类型的控件，也可以放置布局容器（layoutContainer）。

“实例文档 37.xlsm” 中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <backstage>
        <tab id="Tab1" label="group Style" insertBeforeMso="TabShare" columnWidthPercent="40">
            <firstColumn>
                <group id="Group1" label="SendMail" helperText="fill in these fields please.">
                    <primaryItem>
                        <button id="Button2" label="Send Mail" onAction="OA" imageMso="MicrosoftOutlook"/>
                    </primaryItem>
                </group>
                <group id="Group2" label="Detail">
                    <topItems>
                        <labelControl id="Label1" label="you must fill in these fields before send mail"/>
                        <layoutContainer id="Layout1" layoutChildren="horizontal" expand="neither">
                            <editBox id="Edit1" label="To:"/>
                            <editBox id="Edit2" label="Cc:"/>
                        </layoutContainer>
                        <layoutContainer id="Layout2" layoutChildren="vertical" expand="neither">
```

```
                        <editBox id="Edit3" label="Subject:"/>
                        <editBox id="Edit4" label="Body:"/>
                    </layoutContainer>
                </topItems>
            </group>
        </firstColumn>
    </tab>
  </backstage>
</customUI>
```

代码分析：上述 XML 代码创建了一个新的选项卡 “group style”，在其第 1 列中放置了两个 group，分别使用了 primaryItem 和 topItems。

需要注意的是，topItems 中包含一个 labelControl 和 4 个 editBox，其中，To 和 Cc 这两个文本框放置于一个水平排列的布局容器中，Subject 和 Body 文本框一起放置于一个垂直方向的容器中，如图 5-98 所示。

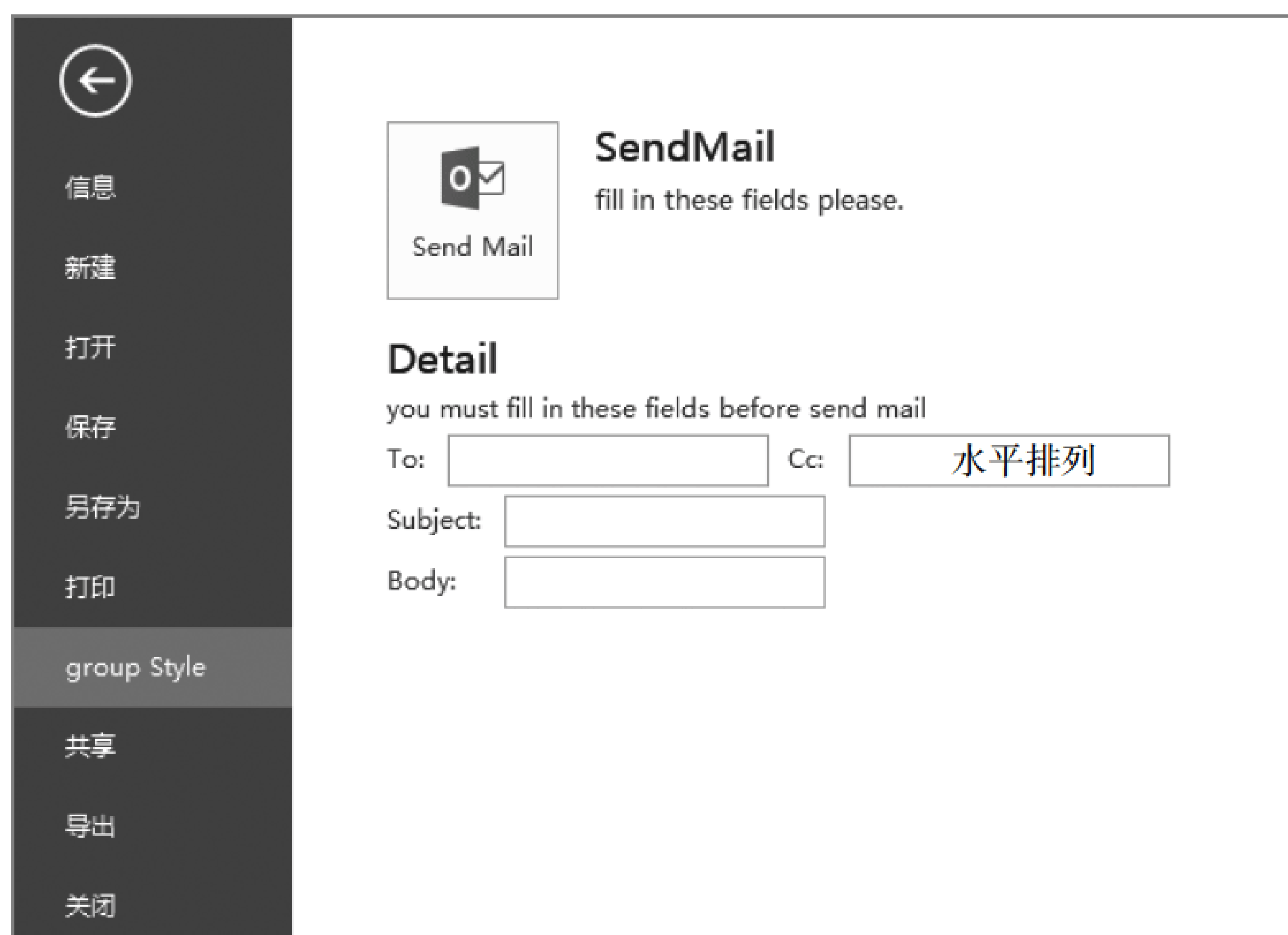


图 5-98　group 风格的 backstage 设计

根据需要，还可以为 tab 增加 secondColumn，这将显示于 firstColumn 的右侧。例如在上述 XML 代码的 firstColumn 节点之后增加如下代码。

```
<secondColumn>
    <group id="Group3" label="ReceiveMail" helperText="click it will receive mails.">
        <primaryItem>
            <button id="Button3" label="Receive Mail" onAction="OA" imageMso=
"MicrosoftOutlook"/>
        </primaryItem>
    </group>
</secondColumn>
```

在 Excel 的 backstage 视图中可以看到新选项卡中多了一列，如图 5-99 所示。

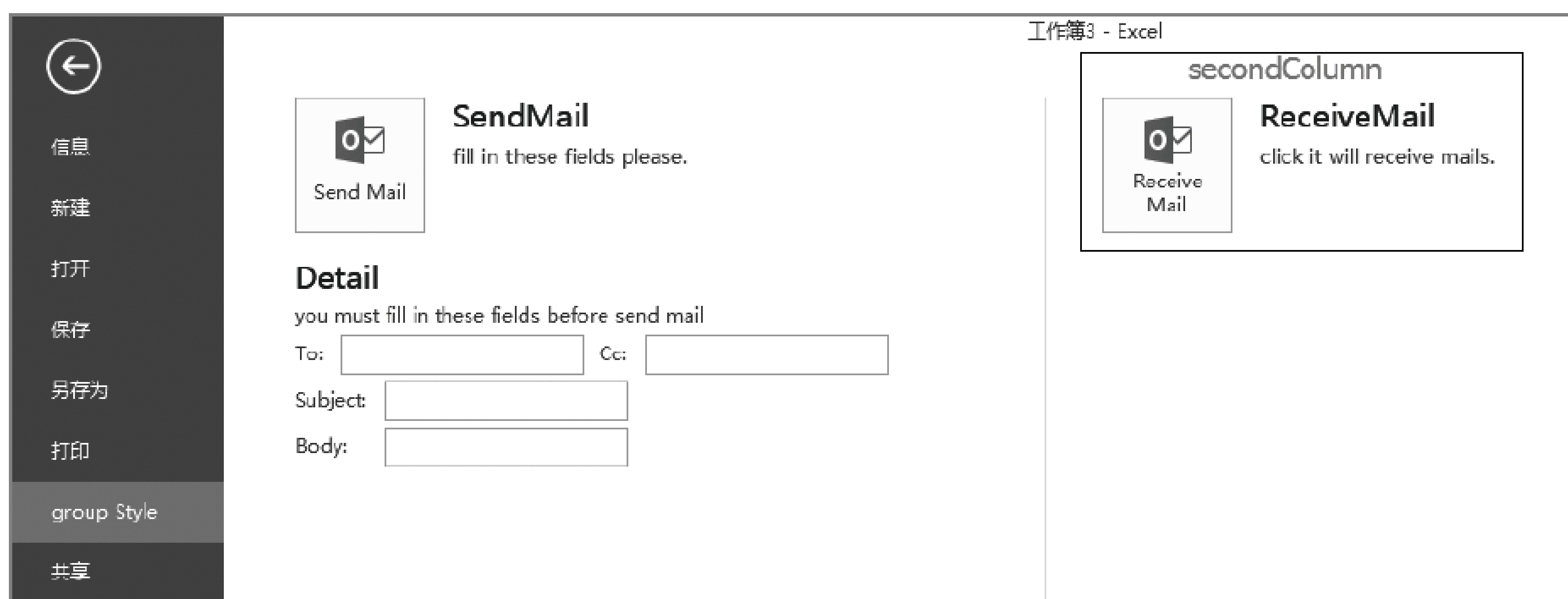


图 5-99 增加 secondColumn

5.9.4 taskGroup 风格

taskGroup 与 group 不同，下面的元素依次是 category 和 task。一个 taskGroup 下面可以有一个以上的 category，一个 category 以下可以有多个 task。

task 和以前学过的 button 控件的用法一样，其中 isDefinitive 属性比较重要，如果设置为 true，表示鼠标单击该 task，会自动退出 backstage 视图，回到 Excel 工作表界面；设置为 false，则还停留在 backstage 视图。

"实例文档 38.xlsm"中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <backstage>
        <tab id="Tab1" label="taskGroup Style" insertBeforeMso="TabShare"
columnWidthPercent="40">
            <firstColumn>
                <taskGroup id="TaskGroup1" label=" 东北地区 ">
                    <category id="Category1" label=" 黑龙江省 ">
                        <task id="Task1"label=" 哈尔滨 " isDefinitive="true"
onAction="OA" imageMso="A"/>
                        <task id="Task2" label=" 佳木斯 " isDefinitive="true"
onAction="OA" imageMso="B"/>
                        <task id="Task3" label=" 大庆 " isDefinitive="false"
onAction="OA" imageMso="C"/>
                    </category>
                    <category id="Category2" label=" 吉林省 ">
                        <task id="Task4" label=" 长春 " isDefinitive="true"
onAction="OA" imageMso="D"/>
                        <task id="Task5" label=" 四平 " isDefinitive="true"
onAction="OA" imageMso="E"/>
                        <task id="Task6" label=" 延边 " isDefinitive="false"
onAction="OA" imageMso="F"/>
                    </category>
                </taskGroup>
            </firstColumn>
        </tab>
    </backstage>
</customUI>
```

当单击“哈尔滨”或“佳木斯”时，会自动返回到工作表界面，而单击“大庆”不自动返回，如图 5-100 所示。

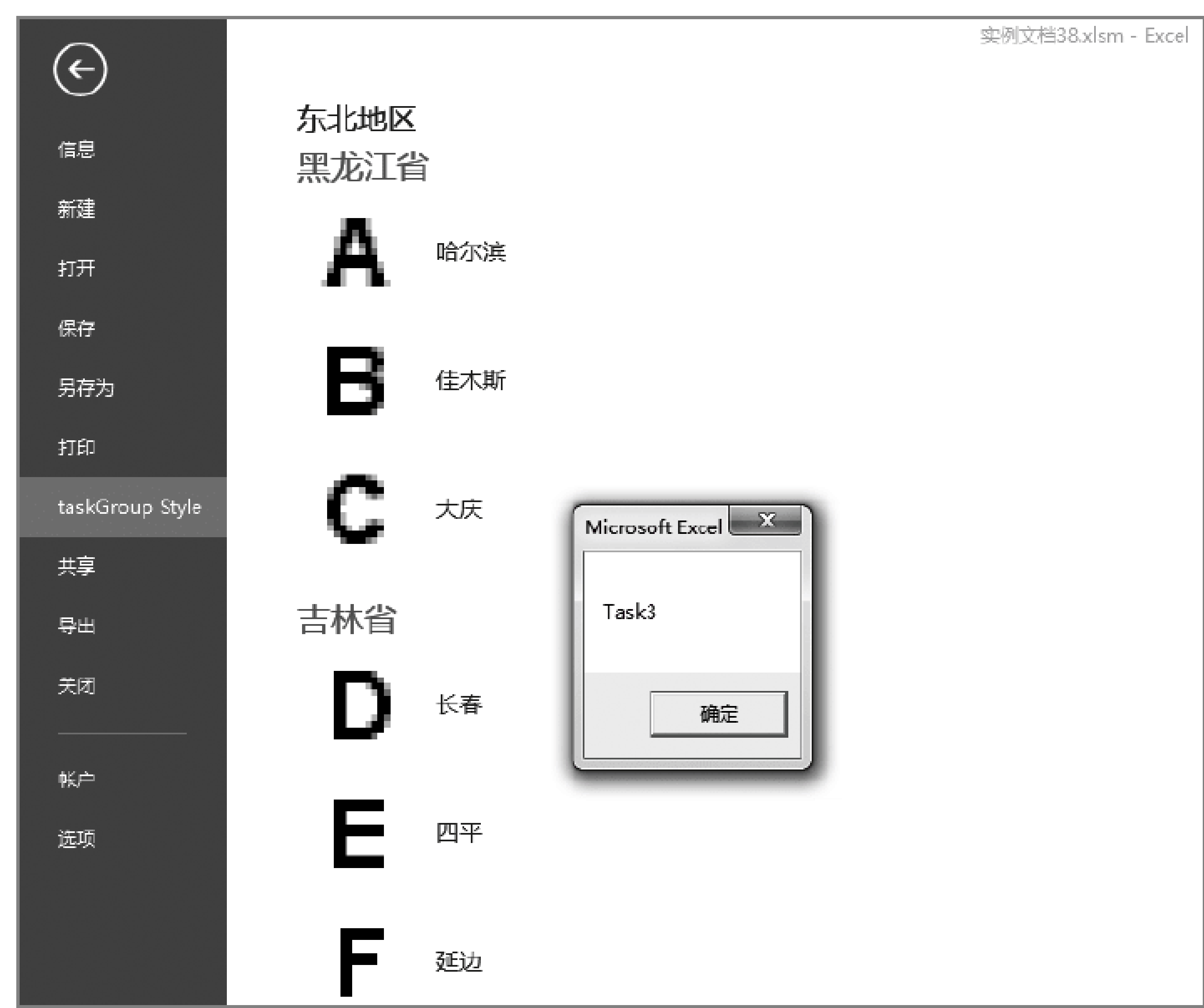


图 5-100　taskGroup 风格

5.9.5　taskFormGroup 风格

taskFormGroup 风格与 taskGroup 风格最为相似，不同的是，taskGroup 风格下面的末梢元素 task 相当于 button 的功能，不能继续展开。而 taskFormGroup 中的 task 元素下面可以继续以 group 为子元素，一个 task 下面可以包含多个 group。因此，使用 taskFromGroup 可以实现级联式的组织结构。例如，Outlook 的邮件列表就是这种机制，单击不同的邮件标题，右侧出现相应的邮件详情。

“实例文档 39.xlsm”中的 customUI 如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <backstage>
        <tab id="Tab1" label="我的简历" insertBeforeMso="TabShare" columnWidth
Percent="40">
            <firstColumn>
                <taskFormGroup id="TaskFormGroup1" label="简历中心" allowedTask
Sizes="largeMediumSmall" helperText="简历中心">
                    <category id="Category1" label="基本信息">
                        <task id="Task1" label="个人信息">
                            <group id="Group1" label="个人信息">
                                <topItems>
                                    <layoutContainer id="Layout0" layoutChildren=
"vertical" expand="neither">
                                        <imageControl id="Image1" getImage="getPhoto"/>
                                        <editBox id="Edit1" label="姓名"/>
```

```
                    <editBox id="Edit2" label="年龄"/>
                </layoutContainer>
            </topItems>
        </group>
    </task>
    <task id="Task2" label="联系方式">
        <group id="Group2" label="联系方式">
            <topItems>
                <editBox id="Edit3" label="手机"/>
                <editBox id="Edit4" label="邮箱"/>
            </topItems>
        </group>
    </task>
</category>
<category id="Category2" label="教育经历">
    <task id="Task3" label="本科">
        <group id="Group3" label="本科">
            <topItems>
                <editBox id="Edit5" label="院校"/>
                <editBox id="Edit6" label="专业"/>
            </topItems>
        </group>
    </task>
    <task id="Task4" label="硕士">
        <group id="Group4" label="硕士">
            <topItems>
                <editBox id="Edit7" label="院校"/>
                <editBox id="Edit8" label="专业"/>
            </topItems>
        </group>
    </task>
    <task id="Task5" label="博士">
        <group id="Group5" label="博士">
            <topItems>
                <editBox id="Edit9" label="院校"/>
                <editBox id="Edit10" label="专业"/>
            </topItems>
        </group>
    </task>
</category>
<category id="Category3" label="外语水平">
    <task id="Task6" label="英语">
        <group id="Group6" label="英语">
            <topItems>
                <labelControl id="Label1" label="大学英语六级"/>
                 <labelControl id="Label2" label="听说读写能
力比较强"/>
            </topItems>
        </group>
    </task>
    <task id="Task7" label="日语">
```

```
                    <group id="Group7" label=" 日语 ">
                        <topItems>
                            <checkBox id="Check1" label=" 具有多年留学经验 "/>
                            <checkBox id="Check2" label=" 具有国外工作经验 "/>
                        </topItems>
                    </group>
                </task>
            </category>
            <category id="Category4" label=" 编程语言 ">
                <task id="Task8" label="VBA">
                    <group id="Group8" label="VBA">
                        <topItems>
                             <layoutContainer id="Layout1" layoutChildren=
"horizontal" expand="neither">
                                <checkBox id="Check3" label="Excel"/>
                                <checkBox id="Check4" label="PowerPoint"/>
                                <checkBox id="Check5" label="Word"/>
                                <checkBox id="Check6" label="Access"/>
                                <checkBox id="Check7" label="Outlook"/>
                            </layoutContainer>
                        </topItems>
                    </group>
                </task>
                <task id="Task9" label="Python">
                    <group id="Group9" label="Python">
                        <topItems>
                            <layoutContainer id="Layout2" layoutChildren=
"vertical">
                                <hyperlink id="Hyper1" label=" 菜鸟教程 "
target="http://www.runoob.com/python3/python3-tutorial.html"/>
                            </layoutContainer>
                            <dropDown id="Drop1" alignLabel="center"
expand="neither">
                                <item id="Item1" label=" 基础语法 "/>
                                <item id="Item2" label=" 序列对象 "/>
                                <item id="Item3" label=" 选择循环 "/>
                                <item id="Item4" label=" 文件读写 "/>
                            </dropDown>
                        </topItems>
                    </group>
                </task>
            </category>
        </taskFormGroup>
    </firstColumn>
</tab>
</backstage>
</customUI>
```

代码分析：XML 代码虽然看起来很长，但是都遵循 taskFormGroup/category/task/group 的四级结构，当用鼠标单击不同的 task，右侧会跳转到相应的 group 中，如图 5-101 所示。



图 5-101　taskFormGroup 风格

5.9.6　重要属性解释

在 backstage 的 customUI 设计过程中会遇到一些前面未出现过的属性名和属性值，如表 5-4 所示。

表 5-4　backstage 设计中用到的常用属性

元　　素	属　性　名	含义或可取值
backstage	onShow、onHide	当显示、关闭 backstage 视图时触发的回调过程
tab	columnWidthPercent	100 以下的整数，表示 firstColumn 所占的百分比，假如是 30 则表示 firstColumn 和 SecondColumn 宽度之比是 3:7
group	hyperText	提示信息
group	style 或 getStyle	取值为 normal、error、warning，规定一个组的显示风格
layoutContainer	layoutChildren	取值为 horizontal、vertical，规定容器中各个控件的排列方向
layoutContainer	expand	取值为 horizontal、vertical、both、neighbor，规定容器中控件的扩展方向
taskFormGroup	allowedTaskSizes	枚举值，规定是否可以调整 task 的尺寸
task	description	task 的描述信息
imageControl	image、imageMso、getImage	显示一个图片的控件
hyperlink	target	显示一个指定 url 的超链接

在实际开发过程中，结合实际项目，对以上属性进行微调即可。

“实例文档 40.xlsm”演示了一个自动更改图片的功能，每当重新进入 backstage 视图，看到的图片是不一样的。其中的 XML 代码如下。

```
    <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
onLoad="OnLoad">
```

```
    <backstage onShow="OnShow" onHide="OnHide">
        <tab id="Tab1" label=" 自动换图 " insertBeforeMso="TabShare" columnWidthPercent=
"60">
            <firstColumn>
                <group id="Group1" label=" 警告 " style="warning">
                    <topItems>
                        <layoutContainer id="Layout1" layoutChildren="vertical"
expand="vertical">
                            <imageControl id="Image1" getImage="GetImage"/>
                                <editBox id="Edit1" label=" 图片名称 " getText=
"GetName"/>
                                <editBox id="Edit2" label=" 符号序列 " getText=
"GetSymbol"/>
                        </layoutContainer>
                    </topItems>
                </group>
            </firstColumn>
        </tab>
    </backstage>
</customUI>
```

与上述 XML 代码对应的 VBA 回调函数如下。

```
'customUI 回调函数模块
Public R As IRibbonUI, i As Integer

Sub OnShow(contextObject As Object)
    i = i + 1
    R.Invalidate
End Sub
Sub OnHide(contextObject As Object)

End Sub

Public Sub OnLoad(ribbon As Office.IRibbonUI)
    Set R = ribbon
    i = 0
End Sub
Public Sub GetImage(control As Office.IRibbonControl, ByRef image)
    Set image = LoadPicture(ThisWorkbook.Path & "\ 扑克牌 \" & i & ".jpg")
End Sub
Public Sub GetName(control As Office.IRibbonControl, ByRef text)
    text = i & ".jpg"
End Sub
Public Sub GetSymbol(control As Office.IRibbonControl, ByRef text)
    text = String(i, " ● ")
End Sub
```

代码分析：使用了 backstage 的 OnShow 回调，这个与 OnLoad 回调不同，OnLoad 回调打开工作簿后只发生一次，而 OnShow 是每当重新打开 backstage 视图都会触发。

另外，本例中 group 的 style 为 warning，那么该组显示的外观为浅红色，如图 5-102 所示，读者可自行尝试。

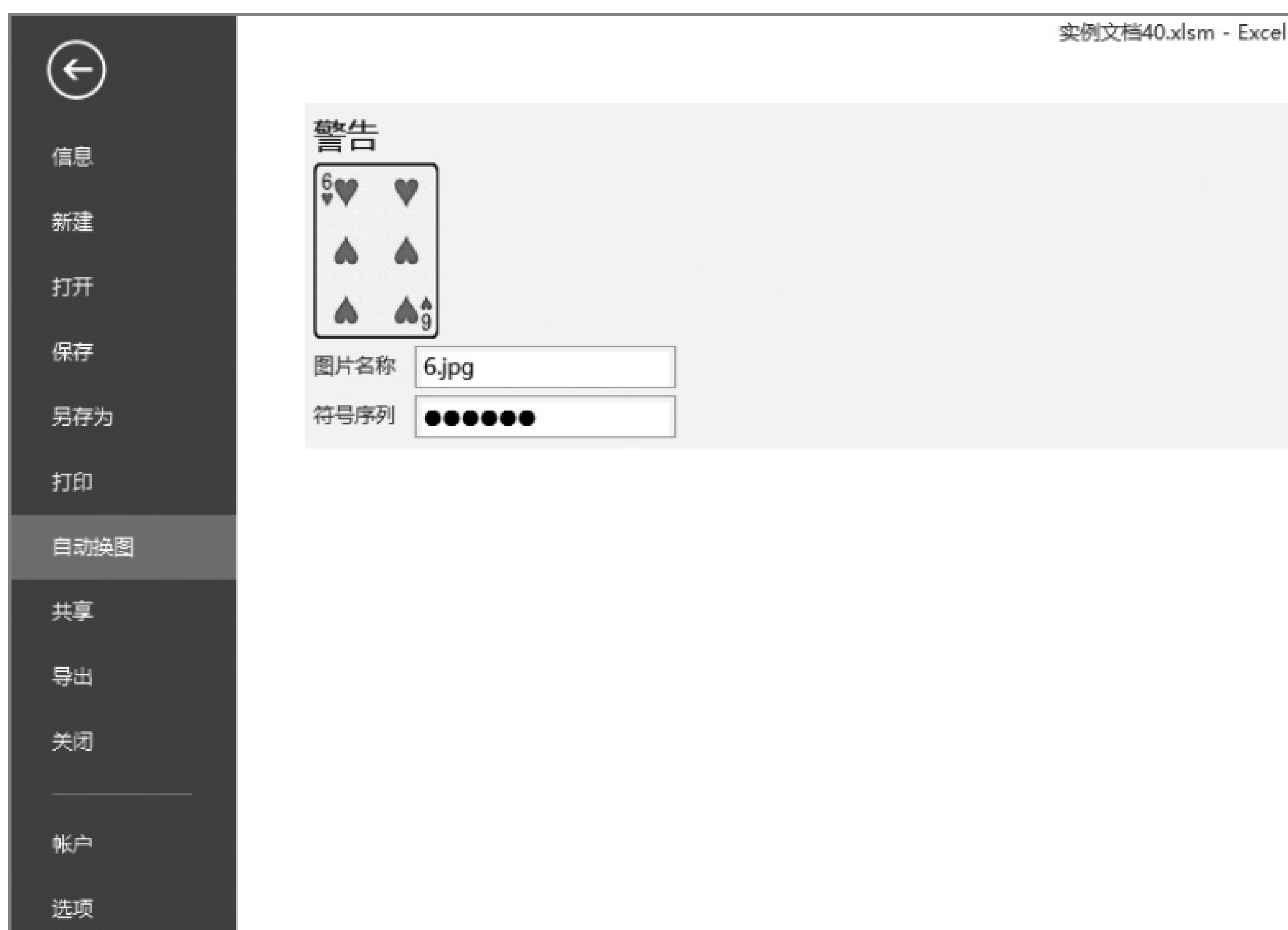


图 5-102 利用 OnShow、OnHide 回调函数动态更新界面

5.10 更改内置控件属性

customUI 除了可以定制前面所述的 5 个场所，还可以使用 commands 重定义内置控件。

commands 元素节点中允许放多个 command，每个 command 节点重新定义一个控件的属性。

允许重新定义的属性主要有 enabled 和 onAction，也就是更改内置控件的可用性和控件的功能。

下面的 XML 代码禁用“保存”功能以及“加粗”按钮。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <commands>
        <command idMso="FileSave" enabled="false"/>
        <command idMso="Bold" enabled="false"/>
    </commands>
</customUI>
```

可以看到 Excel 的保存、加粗都是灰色不可用的，如图 5-103 所示。

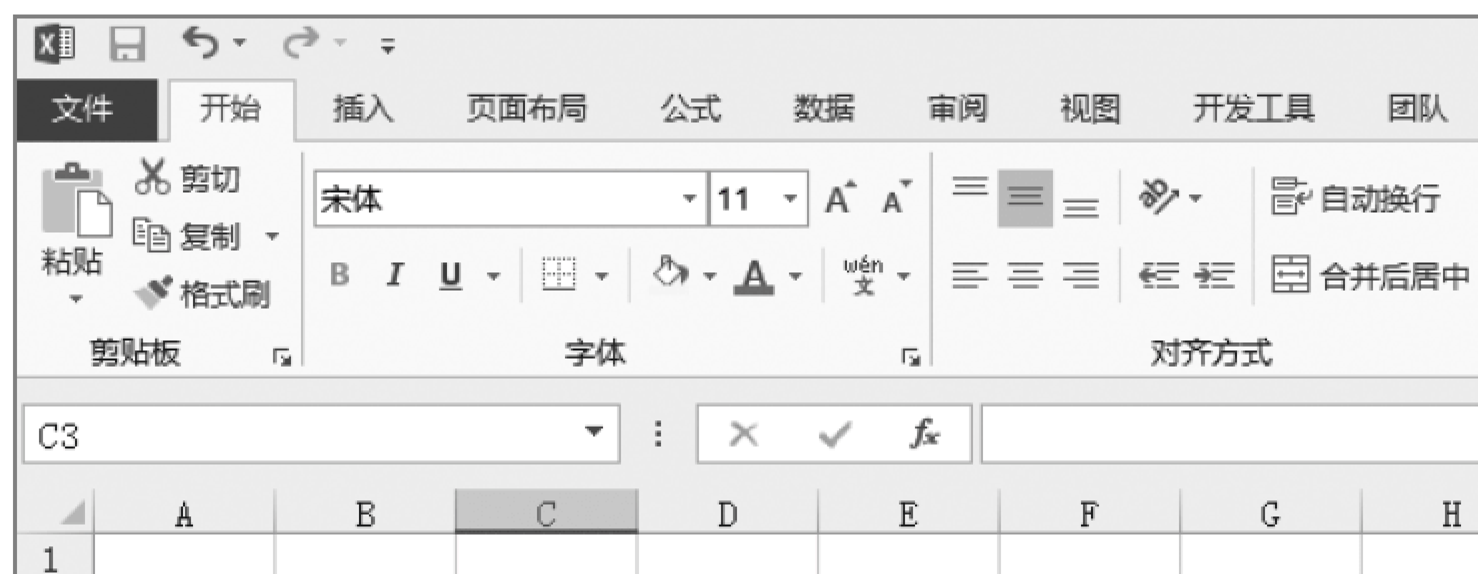


图 5-103 利用 commands 更改内置控件的属性

5.11　customUI 疑难解答

问：能不能隐藏内置组中的控件？能不能任意修改内置组中控件的 label 或者其他属性？

答：不能任意修改内置组中控件的属性。

问：向 Office 文档中压入 XML 代码的技术很重要吗？

答：Office 文档的安全性比较低，容易被破解，因此向 Office 文档中压入 XML，意味着回调函数必须写在文档的 VBA 工程中，分发给他人后，就等于公开文档的一切。因此，XML 如何压入 Office 文档了解一下即可。

问：customUI 呈现出的界面，只有包含该代码的文档处于活动文档才显示界面，怎样变成全局的、不受文档切换的应用程序级的 customUI 呢？

答：对于 Excel，首先把 XML 压入到工作簿文件中，然后把该工作簿另存为 Excel 加载宏（扩展名为 .xlam），加载宏中的 customUI 是全局性的，不随工作簿窗口的切换而改变。

对于 Word，可以把文档另存为扩展名为 .dotm 的模板文件。

对于 PowerPoint，把演示文稿另存为扩展名为 .ppam 的加载宏文件。

问：顺利进行 customUI 开发，具体需要安装哪些工具和软件？

答：主要工具选择 Ribbon XML Editor 或 CustomUI Editor 之一，idMso 查询工具选择 OfficeidMsoViewer，imageMso 查询工具选择 ImageMso7345.xlsm。

对 customUI 的 XML 语法和成员不太熟悉的读者，还可以在 Visual Studio 中快速编辑 XML。

问：能否在同一个文档的 customUI 代码中同时包含自定义多个场所？

答：可以。本书中的实例一般一个工作簿中只定义一个场所，实际上根据需要可以同时定制多个场所。

例如“实例文档 35.xlsm”中的 XML 代码如下。

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="true">
        <tabs>
            <tab id="Tab1" label=" 常用选项卡 "/>
        </tabs>
        <qat>
            <sharedControls>
                <button id="Button1" label=" 按钮 " imageMso="Q"/>
            </sharedControls>
        </qat>
        <contextualTabs>
            <tabSet idMso="TabSetSmartArtTools">
                <tab idMso="TabSmartArtToolsDesign" label=" 我的设计 "/>
            </tabSet>
        </contextualTabs>
    </ribbon>
    <backstage>
        <button id="Button2" label=" 按钮 "/>
```

```
    </backstage>
    <contextMenus>
        <contextMenu idMso="ContextMenuWorkbookPly">
            <button idMso="SheetInsertPage" label=" 插入工作表 "/>
        </contextMenu>
    </contextMenus>
</customUI>
```

以上 customUI 代码同时定制了常用功能区、快速访问工具栏、环境功能区、右键菜单、Office 菜单 5 个场所。

问：Office 2007 的 customUI 定制有哪些注意事项？

答：一定要考虑 customUI 命名空间的影响，如果文档要在 Office 2007 中打开，那么该文档中 customUI 的命名空间必须是：<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">，否则自定义界面无法显示。

如果文档要在 Office 2010 及其以上版本中打开，命名空间最好是<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">，当然也可以是<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">。

实际上，一个 Office 文档可以同时压入两份不同的 XML 代码，从而达到在 Office 2007 中打开和在 Office 2010 中打开呈现不同的界面。

“实例文档 36.xlsm”中包含了 2007 和 2010 两部分的 customUI，在不同版本的 Office 中打开后效果不一样。

5.12 本章小结

使用 customUI 技术，可以在 Office 的很多场所添加用户自定义的界面元素，也可以修改 Office 内置元素。

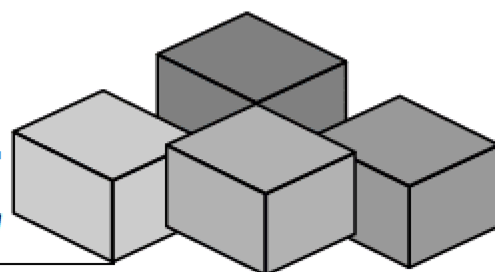
对于常用功能区，tab、group、button 以及其他控件形成了三级 XML 结构。

button 是最常用的元素，常用属性有 label 和 onAction。

对于 customUI 中出现的元素，如果引用的是内置元素，需要指定 idMso 属性，如果是自定义元素，需要指定 id 属性。

第 6 章

使用正则表达式



尽管 VBA 中已经有很多用于字符串处理的函数，例如 Like、Replace、Instr，但是在实际编程过程中，以上函数还是不够简单，例如使用 Like 验证一个字符串是不是恰好为一个手机号，需要写成 Like "[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]"，看起来特别冗长。再例如，验证一个字符串中是否只有小写英文字母，需要在循环中使用 Mid 函数提取出每一个字符。

```
Sub Test()
    Dim i As Integer, c As String
    Const words As String = "Excel 2013"
    For i = 1 To Len(words)
        c = Mid(words, i, 1)
        If (c >= "a" And c <= "z") Then

        Else
            MsgBox "不全是小写字母", vbExclamation
        End If
    Next i
End Sub
```

可以看出，使用 VBA 内置字符串函数，即使处理很简单的一个任务，也需要大量代码，而且很多情况下必须使用循环结构。

正则表达式（Regular Expression，简称 RegExp），使用一个模式来表达和规范某一类字符串，从而可以快速从源文本中进行验证、替换和查找操作。实际上，很多种编程语言都可以使用正则表达式，本章将介绍 VBA 编程中如何使用正则表达式。

本章用到的外部引用和重要对象：

- ❑ Microsoft VBScript Regular Expressions 5.5
    - ➢ VBScript_RegExp_55.RegExp

6.1 正则表达式入门

正则表达式的作用就是处理字符串。学习 VBA 中的正则表达式，重点学习模式的构造以及验证、替换和查找三大方法。

VBA 编程中使用正则表达式的流程如图 6-1 所示。

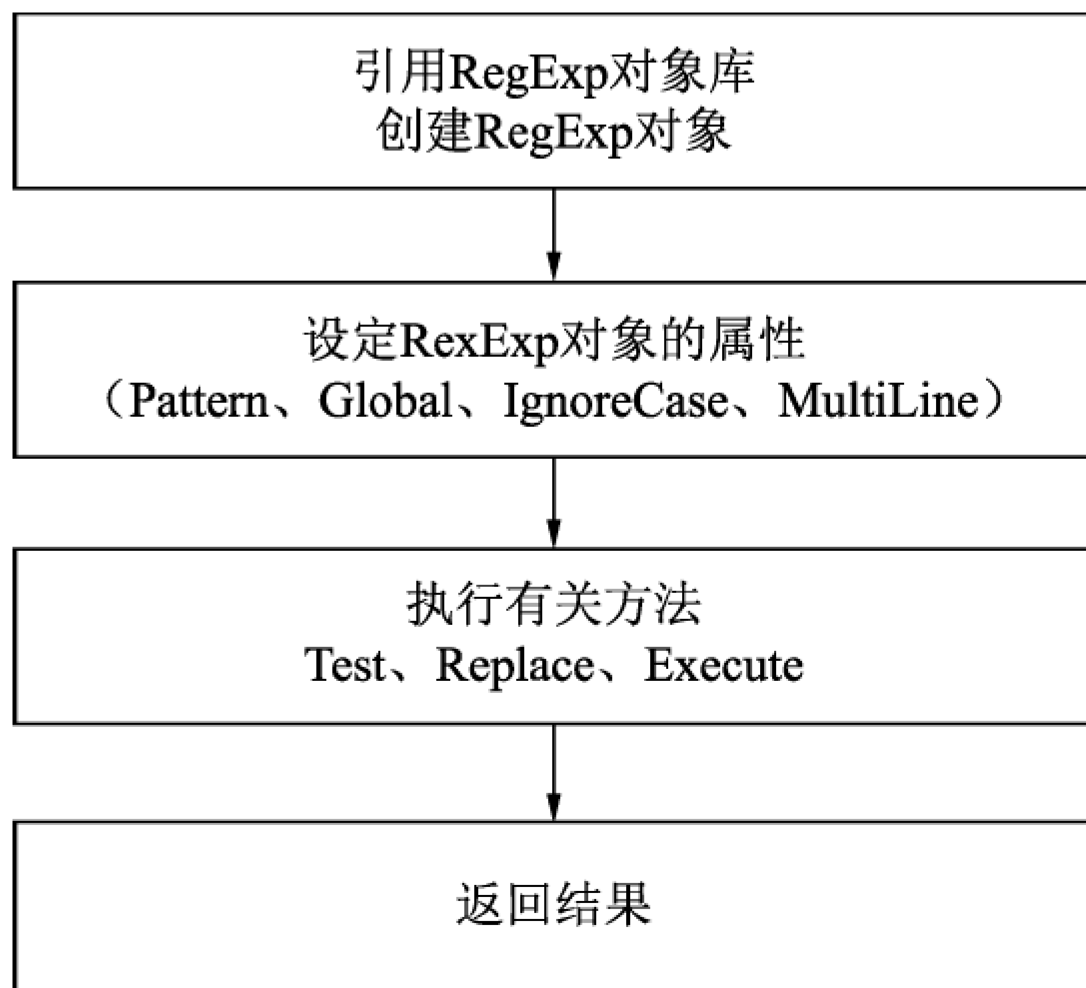


图 6-1 正则表达式的使用流程

6.1.1 引用 RegExp

前期绑定：在 VBA 编辑器中单击【工具 / 引用】，勾选“Microsoft VBScript Regular Expressions 5.5”，如图 6-2 所示。

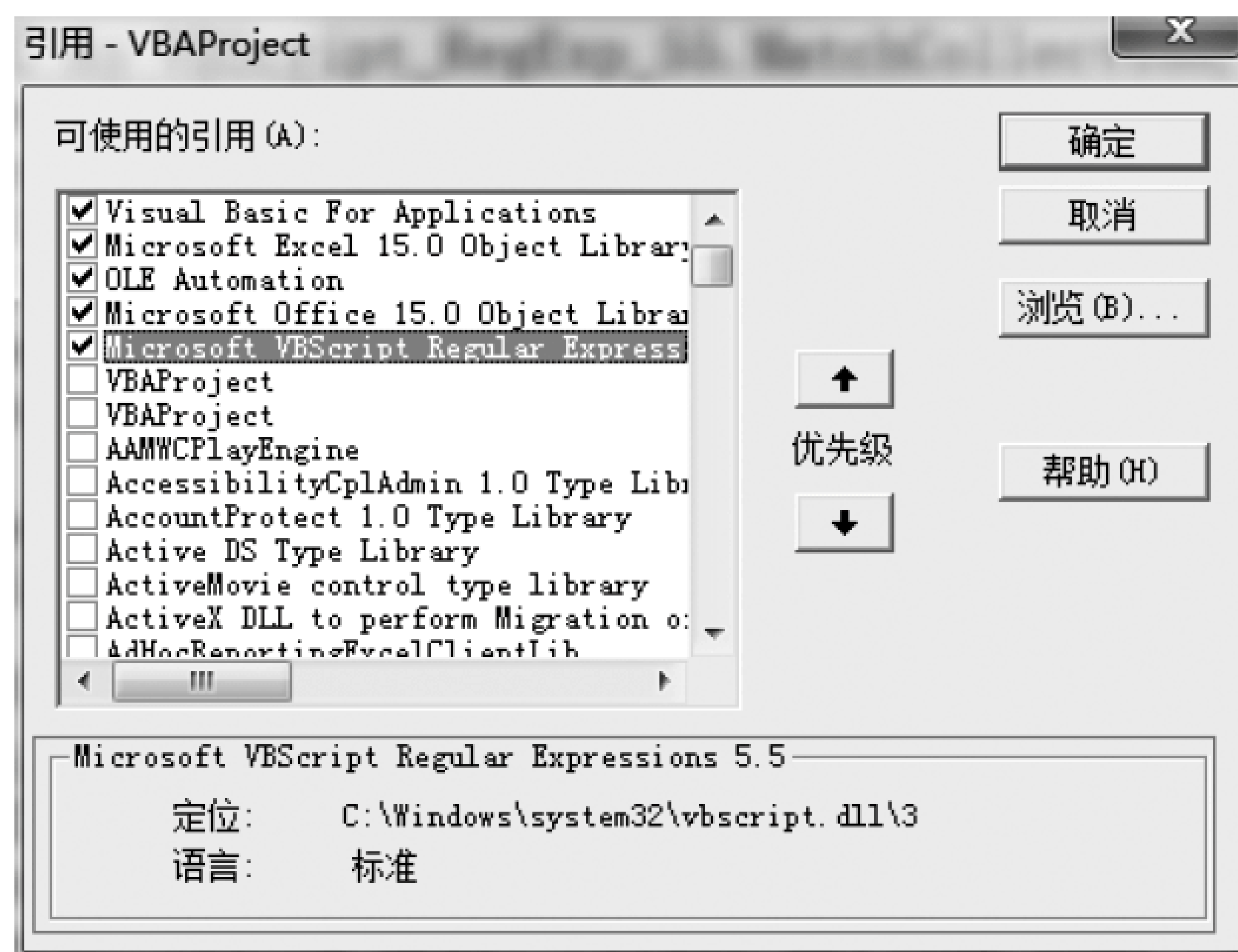


图 6-2 添加外部引用

后期创建对象。

```
Set reg = CreateObject("VBScript.RegExp")
```

本章按照前期绑定方式讲解。

6.1.2　创建 Regexp 对象

在 VBA 代码中声明一个 RegExp 对象，输入小数点后可以看到包含 4 个属性、3 个方法，如图 6-3 所示。

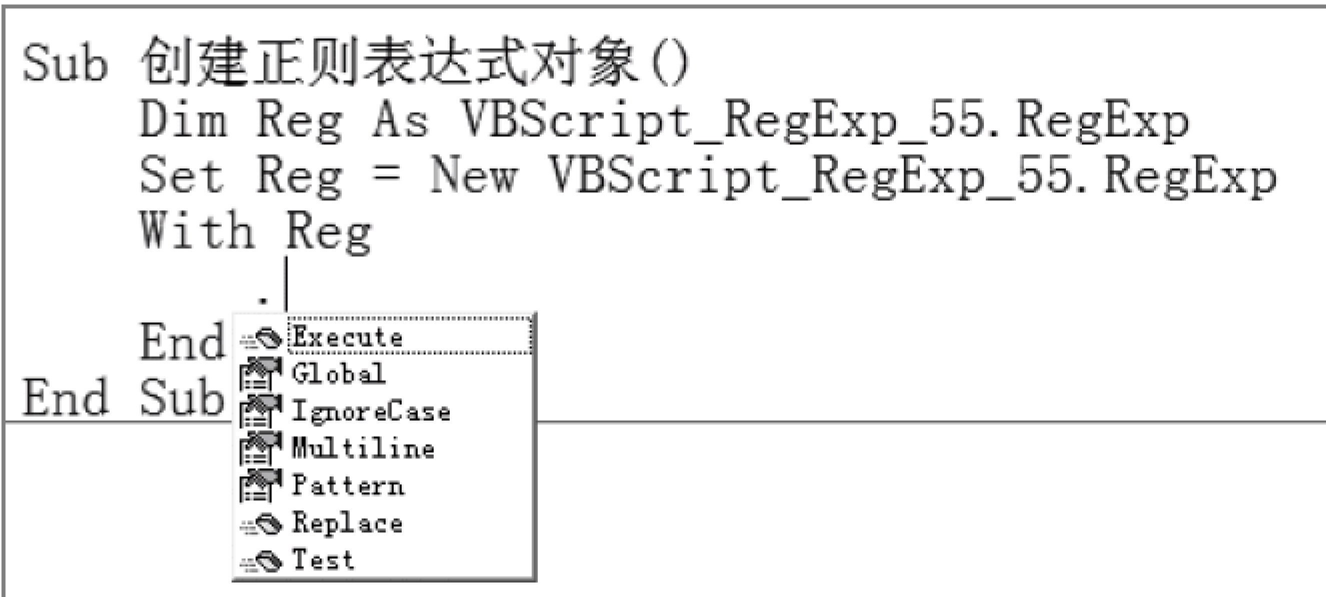


图 6-3　创建正则表达式对象

因此，正则表达式的编程技巧，就在于 Pattern 的构造和各个方法的灵活运用。

正则表达式的 4 个属性中，模式（Pattern）是最主要属性，其他属性是对 Pattern 的补充修饰。组成 Pattern 的基本单位是元字符，元字符的理解掌握，是正则表达式运用的重点和难点。

6.1.3　模式和元字符

在微软 Word 的查找文本框中输入的内容，就相当于正则表达式中的模式（pattern），换句话说，pattern 就是查找目标。

组成 pattern 的字符，既可以是确切的字符，也可以是模糊的字符（元字符），元字符通常可以表达一系列属于同类别的多种字符，如表 6-1 所示。

表 6-1　正则表达式元字符

字　符	描　述
\	转义字符标志
^	匹配输入字符串的开始位置
$	匹配输入字符串的结束位置
*	匹配前面的零次或多次的子表达式
+	匹配前面的一次或多次的子表达式
?	匹配前面的零次或一次的子表达式
{n}	n 是一个非负整数，匹配前面的 n 次子表达式
{n,}	n 是一个非负整数，至少匹配前面的 n 次子表达式
{n,m}	m 和 n 均为非负整数，其中 n ≤ m，最少匹配 n 次且最多匹配 m 次
?	当该字符紧跟在其他限制符（*，+，?，{n}，{n,}，{n，m}）后面时，匹配模式尽可能少地匹配所搜索的字符串

续表

字　符	描　述
.	匹配除“\n”之外的任何单个字符
(pattern)	匹配 pattern 并获取这一匹配结果
(?:pattern)	匹配 pattern 但不获取匹配结果
(?=pattern)	正向预查，在任何匹配 pattern 的字符串开始处匹配查找字符串
(?!pattern)	负向预查，在任何不匹配 pattern 的字符串开始处匹配查找字符串
x|y	匹配 x 或 y。例如，'z|food' 能匹配“z”或“food”；'(z|f)ood' 则匹配“zood”或“food”
[xyz]	字符集合。匹配所包含的任意一个字符。例如，'[abc]' 可以匹配“plain”中的“a”
[^xyz]	负值字符集合。匹配未包含的任意字符。例如，'[^abc]' 可以匹配“plain”中的“p”
[a-z]	匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符
[^a-z]	匹配不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配不在 'a' ~ 'z' 内的任意字符
\b	匹配一个单词边界，指单词和空格间的位置
\B	匹配非单词边界
\d	匹配一个数字字符，等价于 [0-9]
\D	匹配一个非数字字符，等价于 [^0-9]
\f	匹配一个换页符
\n	匹配一个换行符
\r	匹配一个回车符
\s	匹配任何空白字符，包括空格、制表符、换页符等

补充说明：

由于在正则表达式中“\”“?”“*”“^”“$”“+”“(”“)”“|”“{”“[”等字符已经具有一定特殊意义，如果需要用它们的原始意义，则应该对它进行转义，例如希望在字符串中至少有一个“\”，那么正则表达式应该这么写：\\+。

例如 Pattern="A[1-9]"，可以匹配 A3、A8，但不可以匹配 A0，也不可以匹配 B1。这是因为 Pattern 中的 A 是确切字符，[1-9] 是模糊字符，这个模糊字符可以匹配到 1 ~ 9 中的任何一个数字。

6.1.4　是否忽略大小写

正则表达式对象的 IgnoreCase 属性用来设置是否忽略大小写，这是一个可以读写的布尔值，对于新创建的 RegExp 对象，其默认的 IgnoreCase 属性为 False，也就是说默认严格区分大小写。

假定源文本是 "Our Computer"，Pattern="[a-z]+"，当 IgnoreCase 属性为 False 时，匹配到 ur 和 omputer；当 IgnoreCase 属性设置为 True 时，可以匹配到 Our 和 Computer 两个单词。因为 "[a-z]+" 表示连续多个小写字母，当 IgnoreCase 为 True 时，大写字母也能匹配到。

6.1.5　是否多行模式

正则表达式对象的 MultiLine 属性用来设置是否为多行模式，默认值为 False。该属性主要影响 ^ 和 $ 的作用。

当 MultiLine 属性为 False 时，关闭多行模式，此时 ^ 只能匹配字符串的起始，$ 只能匹配字符串的结尾。

当 MultiLine 属性为 True 时，开启多行模式，此时 ^ 既能匹配字符串的起始，也能匹配行的起始；$ 既能匹配字符串的结尾，也能匹配行的结尾。

6.1.6　是否全局搜索

正则表达式对象的 Global 属性用来设置是否全局搜索，默认值为 False。

当 Global 为 False 时，最多只搜索到一处。当 Global 为 True 时，在源文本中查找全部。

假设源文本为："胡萝卜 35 公斤土豆 231 公斤西红柿 24 公斤白菜 1234 公斤"，Pattern="\d+"，当 Global 为 False 时，只找到 35，如果 Gloal 为 True，还可以找到后面的三个数字。

Global 属性主要影响 Replace 和 Execute 方法。

6.2　格式验证测试

正则表达式对象的 Test 方法用来判断源文本是否符合模式规定，或者判断能否查找到一处以上，如果能匹配到，则返回 True，否则返回 False。

Test 方法通常用于判断手机号码是否合法、用户名是否合法等。

6.2.1　判断是否包含特定的字符

使用 Test 方法时，当 Pattern 匹配到源文本中至少一处时，返回 True。Test 的用法是，首先创建正则表达式对象，然后设置其 Pattern，最后向 Test 方法中传递源文本作为参数，Test 方法返回一个布尔值。

下面的过程测试源文本中是否包含 6 个以上连续的数字。

```
Sub Find()
    Dim reg As New RegExp
    With reg
        .Pattern = "\d{6,}"
        MsgBox .Test("2018 年 ")
    End With
End Sub
```

结果返回 False，因为源文本中找不到连续的 6 个以上数字。

下面的过程则可以对发帖内容进行敏感词汇分析，如果在发帖内容中找到至少一个敏感词汇，就弹出警告对话框。

```
Sub 验证是否包含特定词汇 ()
    Dim 帖子内容 As String
    Dim reg As New VBScript_RegExp_55.RegExp
    Dim result As Boolean
    帖子内容 = " 我制作了一个秒杀所有 Excel 操作的工具！ "
    With reg
        .Pattern = " 枪手 | 秒杀 | 屌丝 "
        result = .Test( 帖子内容 )
        If result Then
            MsgBox " 含有敏感词汇！ ", vbCritical
        End If
    End With
End Sub
```

上述程序的运行结果如图 6-4 所示。



图 6-4　验证源字符串中是否有特定词语

6.2.2　判断源文本中是否只包含模式

Test 方法还可以用于验证源文本是否恰好为模式，例如验证手机号格式是否正确，或者论坛注册的用户名是否符合规定。

手机号一共包含 11 位数字，其中第 1 位必须是 1，第 2 位是 3、4、5、7、8 之一，剩余的 9 个数字则是任意数字即可。因此可以构造 Pattern 为 "^1[3,4,5,7,8][0-9]{9}$"，前面加 ^ 表示以 1 开头的，后面加 $ 表示以 9 个数字结尾的，加上这两个符号就可以实现恰好匹配。

```
Sub 验证是否为有效手机号 ()
    Dim phone As String
    Dim reg As New VBScript_RegExp_55.RegExp
    Dim result As Boolean
    phone = "138123456789"
    With reg
        .Pattern = "^1[3,4,5,7,8][0-9]{9}$"
        result = .Test(phone)
        If result Then
            MsgBox " 号码有效 ", vbInformation
        Else
            MsgBox " 无效号码 ", vbExclamation
        End If
    End With
End Sub
```

上述过程中，由于多了一个数字，也就是说该号码不是以 9 个连续数字作为结尾，因此弹出“无效号码”对话框。

下面的实例在 VBA 的用户窗体上模拟用户名的注册，假设该论坛允许的用户名只能由数字、字母、下画线组成，并且必须是 4 ～ 8 个字符。

在文本框的 Exit 事件代码中，当鼠标指针试图离开文本框时进行模式验证。

```
Private Sub TextBox1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim reg As New VBScript_RegExp_55.RegExp
```

```
    Dim result As Boolean
    With reg
        .IgnoreCase = True
        .Pattern = "^[0-9A-Z_]{4,8}$"
        result = .Test(Me.TextBox1.Text)
        If result = False Then
            MsgBox "用户名无效，只允许 4 ~ 8 位数字、字母、下画线！ ", vbExclamation
        End If
    End With
End Sub
```

由于用户名允许大写和小写英文字母，因此设置 IgnoreCase 为 True。启动窗体后，输入带有汉字的用户名时，弹出警告对话框，如图 6-5 所示。

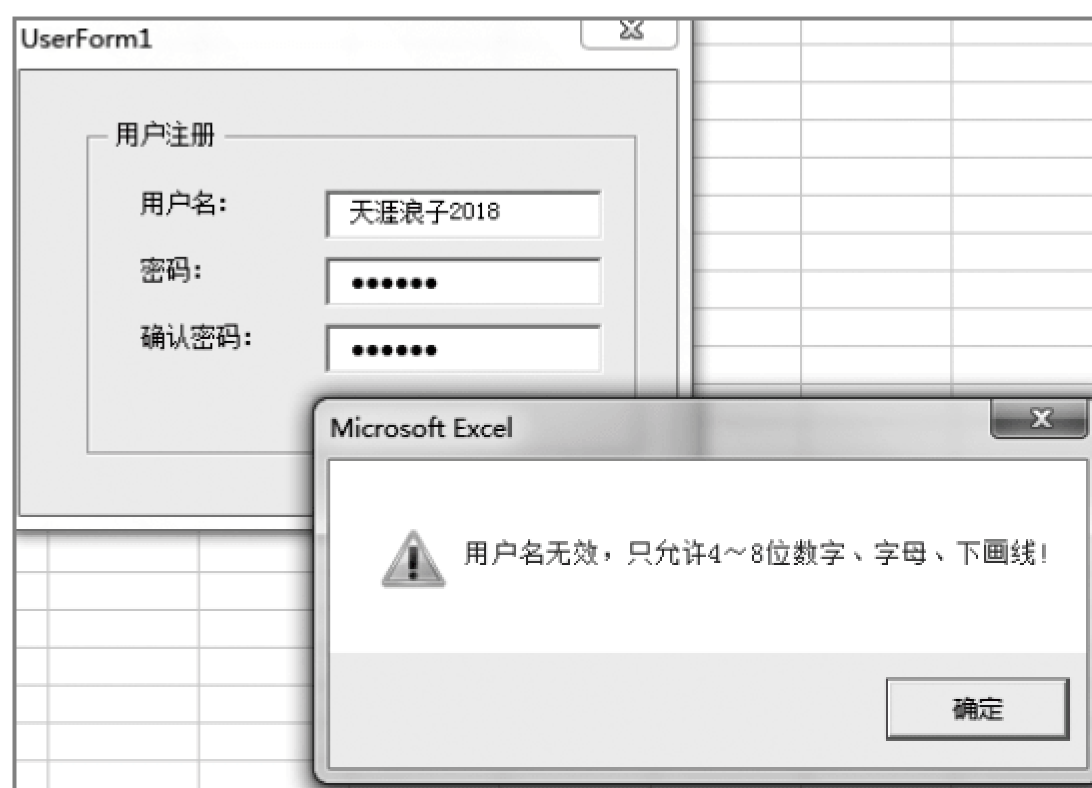


图 6-5　自动判断用户输入的内容是否符合格式要求

6.3　替换

正则表达式的 Replace 方法可以把根据 Pattern 匹配到的目标替换为其他字符串。替换后，将产生一个新字符串，但是并不会破坏源文本。

Replace 方法的语法如下。

```
Result=RegExp.Replace(Source,Replacement)
```

下面的过程演示了 3 个不同的替换过程。

```
Sub 替换数字 ()
    Dim Source As String
    Dim reg As New VBScript_RegExp_55.RegExp
    Dim result As String
    Source = "2002 年的第 1 场雪 "
    With reg
        .Pattern = "\d+"
        result = .Replace(Source, "#")
        Debug.Print result
        .Pattern = "\d"
        result = .Replace(Source, "#")
        Debug.Print result
        .Pattern = "\d"
```

```
            .Global = True
            result = .Replace(Source, "#")
            Debug.Print result
        End With
    End Sub
```

代码分析：上述过程先后进行了三次替换，第 1 次的 Pattern 是 \d+，表示连续的多个数字替换为 1 个 #。第 2 次的 Pattern 是 \d，表示 1 个数字就替换为 1 个 #。但由于前两次 RegExp 的 Global 属性均为 False，只能是第一个匹配项被执行到替换，后面不变。

第 3 次的 Pattern 依然是 \d，但是设置 Global 为 True，意味着是全局操作。

运行上述过程，三次替换的结果如图 6-6 所示。

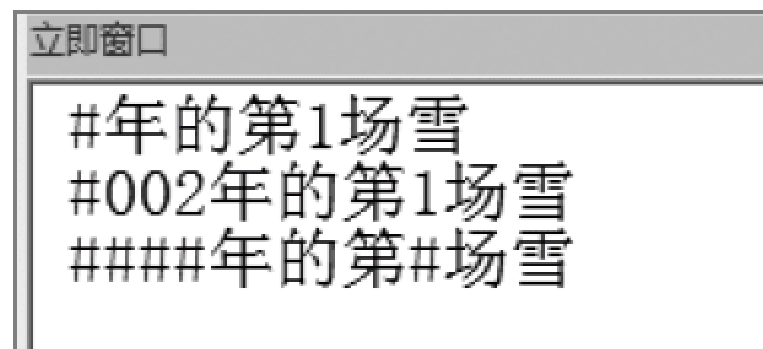


图 6-6　替换的结果

正则表达式中的 + 表示贪婪匹配，把尽可能多的字符当作一个匹配项。Global 属性为 True 时，表示全部替换。

另外，RegExp 的 IgnoreCase 属性也会影响到 Replace 的结果。

下面的过程把英文字母以外的其他字符替换为空字符串，也就是删除非英文字符。

```
    Sub 替换字母以外的所有字符 ()
        Dim Source As String
        Dim reg As New VBScript_RegExp_55.RegExp
        Dim result As String
        Source = "Happy New Year 2018"
        With reg
            .Global = True
            .Pattern = "[^A-Z]"
            result = .Replace(Source, "")
            Debug.Print result
        End With
    End Sub
```

代码分析：[^A-Z] 表示 A ~ Z 之外的所有字符，方括号内的 ^ 表示范围之外的意思。由于没有设置 RegExp 的 IgnoreCase 属性，所以默认按照区分大小写处理，运行的结果是 HNY，发现删除了所有的小写英文字母。

如果在上述代码中插入一行：.IgnoreCase = True，再次运行，结果为 HappyNewYear。以上就是 IgnoreCase 属性对替换结果的影响。

一般情况下，Replace 方法的 Replacement 与被替换的内容毫无关系，但在某些场合下，还可以再次利用匹配到的目标作为 Replacement 的一部分。

下面的程序把每一个城市的区号转移到电话号码的后面。

```
    Sub 替换包含自身 ()
        Dim Source As String
        Dim reg As New VBScript_RegExp_55.RegExp
        Dim result As String
         Source = "北京：010-12345678，沈阳：024-23456789，呼和浩特：0471-3958123，武汉：
027-87654321"
```

```
    With reg
        .Global = True
        .IgnoreCase = True
        .Pattern = "(\d+)\-(\d+)"
        result = .Replace(Source, "$2@$1")
        Debug.Print result
    End With
End Sub
```

代码分析：上述代码中的 Pattern 中有两个圆括号，分别把连续的数字括住，因此第 1 个圆括号构成了一个分组，那么在 Replace 方法中的 $1 就代表第 1 分组匹配到的内容，$2 表示第 2 个分组。

运行上述过程的结果为：

```
北京：12345678@010，沈阳：23456789@024，呼和浩特：3958123@0471，武汉：87654321@027
```

以上内容的源代码文件为“实例文档 41.xlsm”。

6.4　查找

正则表达式经常用于从纷繁复杂的源文本中筛选出重要的信息，也就是目标文本的匹配或查找。

随着互联网技术的飞速发展，网页上巨大的信息数据需要提取和重新整合，这就可以用正则表达式从网页显示内容或者从网页源代码进行查找，因此正则表达式和网页抓取技术越来越紧密。

下面是网页上的一段话，其中谈到哪些省份呢？

“2017 年 ×× 大学录取分数线 北京：理科 671 分文科 668 分，福建：理科 649 分文科 637 分，内蒙古：理科 668 分文科 647 分，广西：理科 659 分文科 681 分。”

从源文本中定位到想要的内容，一定要观察和发现目标附近的特点，不难看出，每个省份的后面都有个中文冒号，因此可以用这样的语言来描述：

连续多个汉字后面有个冒号，冒号前面的部分就是省份名称！

所以 Pattern 就可以写作：[一-龥]+：

其中，方括号那部分用来匹配任意一个中文汉字，后面的加号表示连续多个汉字，最后的冒号是一个环境标识，因为源文本中也有冒号。

6.4.1　MatchCollection 对象

正则表达式使用 Execute 方法执行查找，查找后返回一个 MatchCollection 集合对象，该集合对象包含了所有的匹配内容。

下面的过程从源文本中查找所有的省份名称。

```
Sub 查找全部 ()
    Dim Reg As New VBScript_RegExp_55.regexp
    Dim MC As VBScript_RegExp_55.MatchCollection
    Dim M As VBScript_RegExp_55.Match
    Dim Source As String
     Source = "2017 年 ×× 大学录取分数线  北京：理科 671 分文科 668 分，福建：理科 649 分文科
637 分，内蒙古：理科 668 分文科 647 分，广西：理科 659 分文科 681 分 "
    With Reg
        .Global = True
        .Pattern = "[ 一 - 龥 ]+： "
        Set MC = .Execute(Source)
        Debug.Print MC.Count
        Set M = MC.Item(0)
        Debug.Print M.Value
    End With
End Sub
```

代码分析：RegExp 的 Execute 方法必须以源文本为参数，该方法返回一个 MatchCollection，如果找到目标，那么该对象的 Count 属性大于 0，否则等于 0。另外，一般情况下执行 Execute 方法时，往往要把 RegExp 的 Global 属性设置为 True，否则只能找到第一个匹配项。

运行上述过程，立即窗口中打印出匹配到的个数，以及第 1 个匹配项的值，如图 6-7 所示。

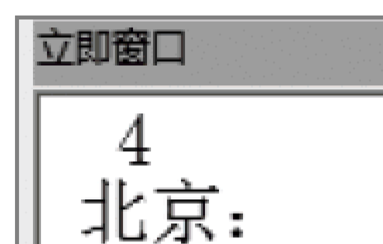


图 6-7　运行结果

6.4.2　Match 对象

Match 对象代表每一个匹配项，恰好是 MatchCollection 的个体对象。此外，Match 匹配项对象有以下 3 个重要属性。

- FirstIndex：匹配项出现在源文本中的位置（最左边是 0）。
- Length：匹配项的长度。
- Value：匹配项的内容。

下面的过程遍历 MatchCollection 中的每一个匹配项。

```
Sub 遍历匹配项 ()
    Dim Reg As New VBScript_RegExp_55.regexp
    Dim MC As VBScript_RegExp_55.MatchCollection
    Dim M As VBScript_RegExp_55.Match
    Dim Source As String
     Source = "2017 年 ×× 大学录取分数线  北京：理科 671 分文科 668 分，福建：理科 649 分文科
637 分，内蒙古：理科 668 分文科 647 分，广西：理科 659 分文科 681 分 "
    With Reg
        .Global = True
        .Pattern = "[ 一 - 龥 ]+： "
        Set MC = .Execute(Source)
        For Each M In MC
            Debug.Print M.FirstIndex, M.Length, M.Value
        Next M
    End With
End Sub
```

运行上述过程，立即窗口打印出每个匹配项的信息，如图 6-8 所示。

立即窗口

15	3	北京：
31	3	福建：
47	4	内蒙古：
65	3	广西：

图 6-8　查找所有匹配项

这里以第 3 行结果为例，数字 47 表示内蒙古出现在第 47 个位置，共 4 个字符。

在实际编程过程中，经常遍历匹配项，把结果发送给 Excel 单元格或数组中以便进一步利用。

实际上，Match 对象的三大属性还存在如下关系。

```
Value=Mid(Source,FirstIndex+1,Length)
```

也就是匹配项的内容可以用 VBA 的 Mid 函数结合 FirstIndex、Length 提取出来。

下面的实例提取中英文混合的文章段落里面的全部英文单词。首先把 MatchCollection 中的每个匹配项转移到字符串数组中，然后再把数组赋给单元格区域。

```
Sub 查找所有单词 ()
    Dim Reg As New VBScript_RegExp_55.regexp
    Dim MC As VBScript_RegExp_55.MatchCollection
    Dim M As VBScript_RegExp_55.Match
    Dim Source As String
    Dim Result() As String
    Dim i As Integer
     Source = "Microsoft Word是文书处理软件，被认为是Office的主要程序，在文字处理软件市场
上拥有垄断份额，其私有的DOC格式被尊为一个行业的标准，虽然由Word 2007年已经转用DOCX格式。Word也
适宜某些版本的Microsoft Works。它适宜Windows和Macintosh平台。它的主要竞争者是LibreOffice、
Corel WordPerfect和ApplePages。"
    With Reg
        .Global = True
        .IgnoreCase = True
        .Pattern = "[a-z]+"
        Set MC = .Execute(Source)
        ReDim Result(0 To MC.Count - 1) As String
        i = 0
        For Each M In MC
            Result(i) = M.Value
            i = i + 1
        Next M
    End With
    Range("A1").Resize(MC.Count).Value = Application.WorksheetFunction.Transpose
(Result)
End Sub
```

代码分析：由于可能存在大写英文字母，所以要把 IgnoreCase 设为 True。在匹配之前不知道有多少个匹配项，因此需要声明一个动态字符串数组，当执行查找后再根据匹配到的个数重新定义数组大小。

i 是一个伴随变量，当遍历匹配项时，自增 1。由于要把一维数组放到一列中，所以最后用到了工作表的转置函数 Transpose。

执行上述代码，Excel 中的 A 列出现所有英文单词，如图 6-9 所示。

	A
1	Microsoft
2	Word
3	Office
4	DOC
5	Word
6	DOCX
7	Word
8	Microsoft
9	Works
10	Windows
11	Macintosh
12	LibreOffice
13	Corel
14	WordPerfect
15	ApplePages
16	

图 6-9　找出所有英文单词

6.4.3　SubMatches 对象

正则表达式允许在 Pattern 中使用圆括号，括起来的内容将形成“子匹配”，或者叫小分组。

还是以高考录取分数线的那段话为例，如果要找的除了省份名称以外，还要把相应的理科和文科分数也提取出来，那需要把 Pattern 修改为：[一-龥]+：理科\d+分文科\d+分。

本实例在正则表达式测试器中的效果如图 6-10 所示。

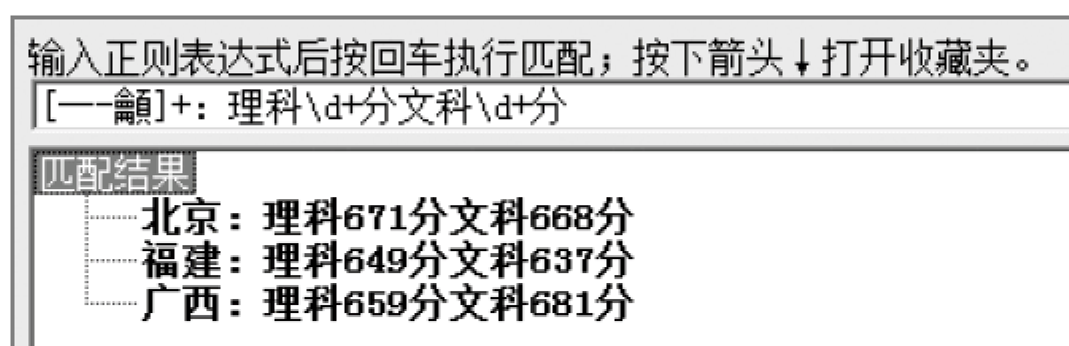


图 6-10　不使用分组的情况

可以看到，匹配到的内容中，冒号没有作用，“理科”“文科”这些常量也没什么用。因此可以在 Pattern 中把想要的重要内容用圆括号括起来以构成分组，然后从小分组中提取。

下面的程序使用分组提取核心内容。

```
Sub 使用分组()
    Dim Reg As New VBScript_RegExp_55.regexp
    Dim MC As VBScript_RegExp_55.MatchCollection
    Dim M As VBScript_RegExp_55.Match
    Dim Source As String
    Source = "2017年××大学录取分数线 北京：理科671分文科668分，福建：理科649分文科637分，内蒙古：理科668分文科647分，广西：理科659分文科681分"
    With Reg
        .Global = True
        .Pattern = "([一-龥]+)：理科(\d+)分文科(\d+)分"
        Set MC = .Execute(Source)
        For Each M In MC
            Debug.Print M.SubMatches(0), M.SubMatches(1), M.SubMatches(2)
        Next M
    End With
End Sub
```

代码分析：Pattern 中通过 3 对圆括号来包括核心内容，那么括号外部的内容就是无用的。对于每一个匹配项，都有 3 个 SubMatches 对象，其中第一个小分组的索引是 0。

运行上述代码，在立即窗口可以看到这段内容的核心数据，如图 6-11 所示。

立即窗口

北京	671	668
福建	649	637
内蒙古	668	647
广西	659	681

图 6-11　使用分组的结果

本实例在正则表达式测试器中的效果如图 6-12 所示。

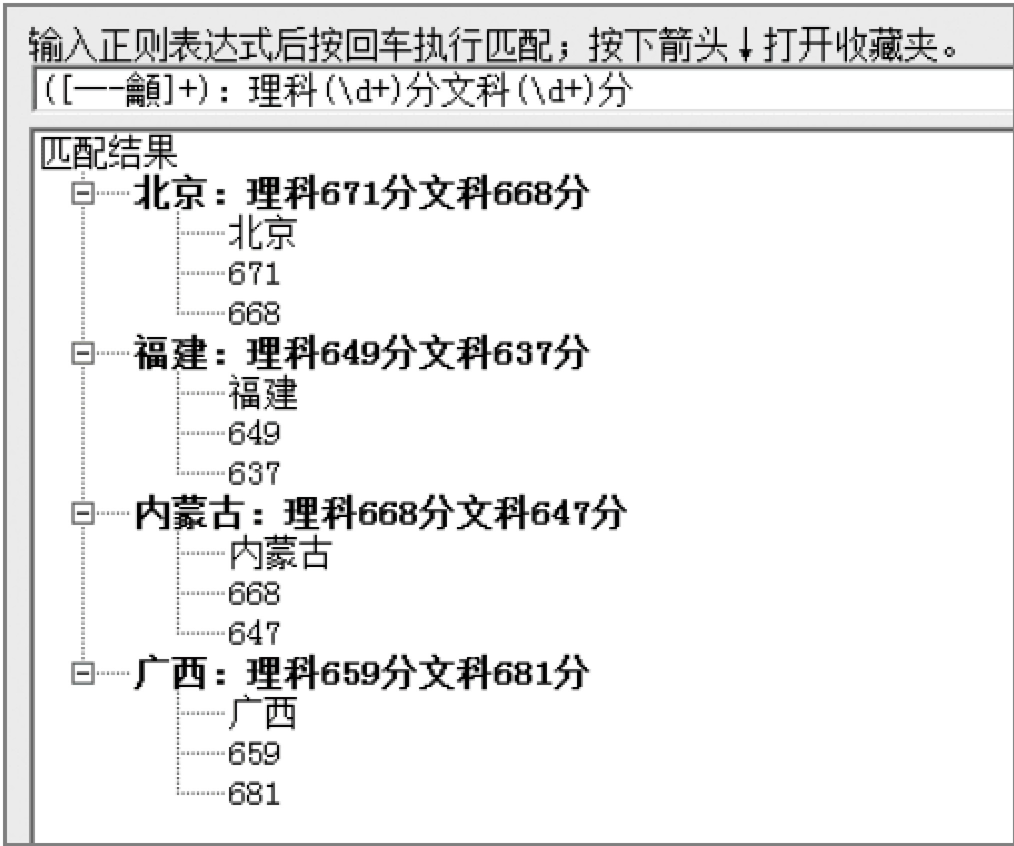


图 6-12　使用分组的情况

正则表达式测试器在后面内容中会有介绍。以上内容的源代码文件为“实例文档 42.xlsm”。

6.5　元字符用法详解

前面虽然讲述了 RegExp 在 VBA 编程中的各种用法，但是真正提高正则表达式应用水平的瓶颈是对元字符的理解。只有很好地理解了各种元字符的含义，才能书写出恰如其分的 Pattern。因此本节总结归纳一下各类元字符的用法和技术点。

6.5.1　字符范围

表示字符范围通常有以下三种方式。

1. 方括号

[bdg]os 可以匹配 bos、dos、gos。

第 [1-8] 名 可以匹配第 3 名，但是不可匹配第 9 名。

[我你他 X-Z] 好 可以匹配我好、你好、他好、X 好、Y 好、Z 好。

需要注意的是，[A-z] 的含义与 [A-Za-z] 不一样，前者可以匹配大小写英文字母外，还可以匹配 ASCII 码值在 91 ～ 96 的字符。后者只能匹配 52 个英文字母。

方括号中最前面加 ^ 表示范围的补集。例如 [^ 你我他] 在吗 可以匹配她在吗，但是不可以匹配你在吗。

2. 转义字符

\d 表示 0 ～ 9 的一个数字，等价于 [0-9]，\D 是 \d 的补集，等价于 [^0-9]。

\s 能匹配任意一个空白字符，例如空格、制表位、回车符、换行符都可以用 \s 匹配到。因此 \sr 的范围要比 \t、\r、\n 大。\S 是 \s 的补集，表示任意一个非空白字符。

\w 能匹配字母、数字、下画线，相当于 [A-Za-z0-9_]，\W 是 \w 的补集。

\b 匹配英文单词边界，例如在句子“ I have a pencil and a pen, What happened ?”中，Pattern 为 pen 时可以匹配到 3 处，但是 Pattern 换为 \bpen\b 只能匹配到中间的一处。因为两边加上 \b 表示两边再没有英文字母。

3. 小数点

正则表达式的 Pattern 中，小数点可以匹配除了换行符（\n）以外的所有类型字符，可以说小数点的表示范围是最大的，经常使用 .+ 表示连续多个任意字符。在 VBA 中，回车符（\r）用 vbCr 生成，换行符（\n）用 vbLf 生成，而 vbCr & vbLf 可以写作 vbNewLine。

6.5.2 多个可选

使用竖杠分隔各个词汇，可以实现多个可选，例如用 (哈密 | 冬 | 老太婆) 瓜 去匹配句子：

“早上吃了冬瓜汤，一个哈密瓜花了很多钱，至于老太婆瓜不太喜欢吃。”

匹配到 3 处，并且形成了小分组，如图 6-13 所示。

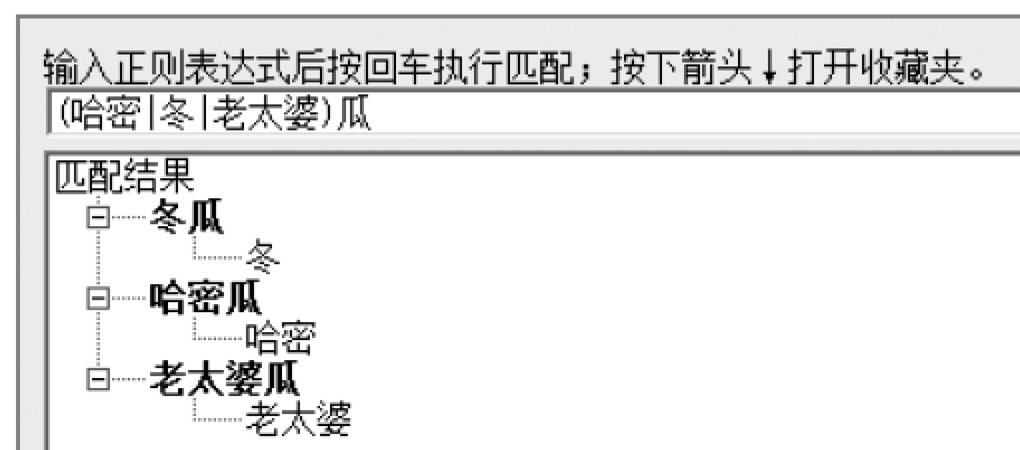


图 6-13 组中的多个词汇之一

如果去掉圆括号，直接用 哈密 | 冬 | 老太婆瓜，匹配结果如图 6-14 所示。

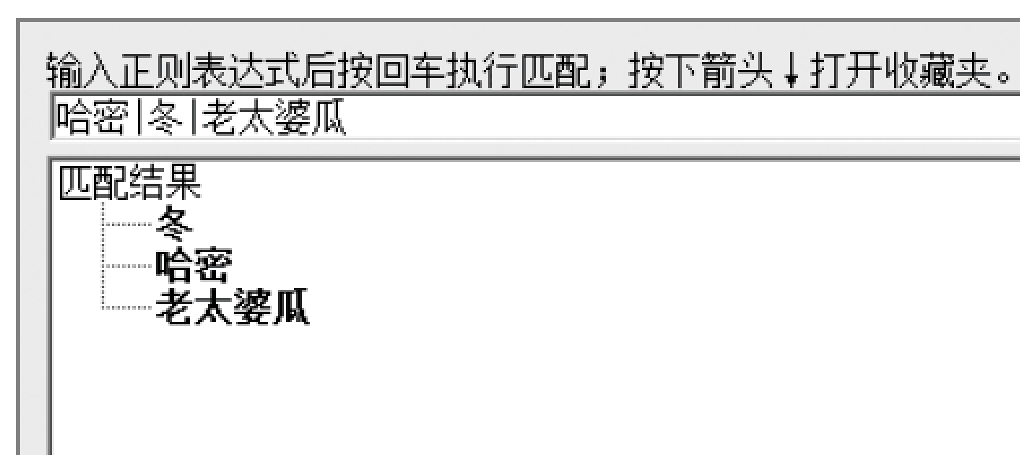


图 6-14 外部的多个词汇之一

6.5.3 环境修饰

使用环境修饰，可以匹配处于某些特定位置的内容，环境修饰符本身不会成为匹配内容，只起到一个参考的作用。常用的环境修饰符如下。

- \b 和 \B：\b 是匹配单词边界，前面已经经过。
- ^ 和 $：^ 匹配行的开头。当 MultiLine 为 True 时，匹配每行的开头位置。$ 匹配行的结尾。

针对下面的 4 行英文歌词，Pattern 为 ^[a-z]+ 时，可以匹配到每行开头的 4 个单词，如图 6-15 所示。

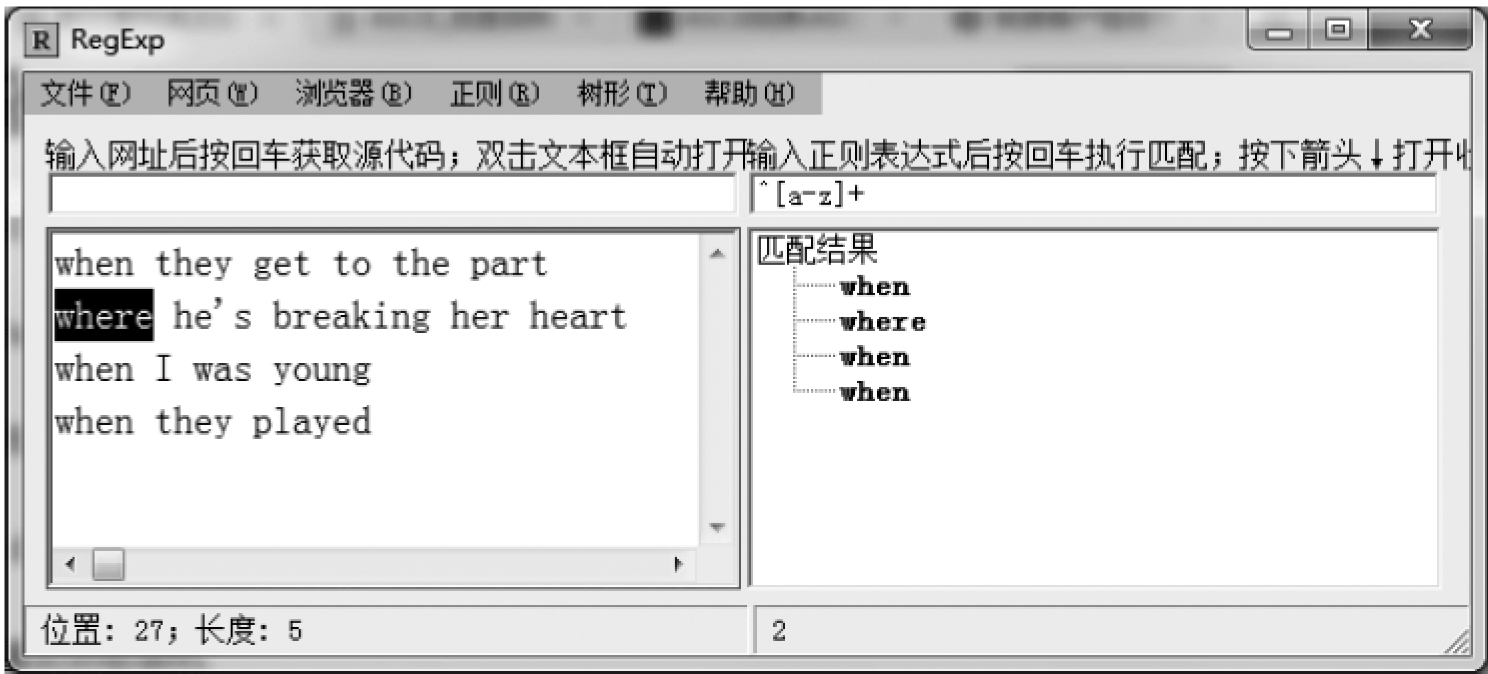


图 6-15　匹配行首

Pattern 为 [a-z]+$ 时，匹配每行尾部的完整单词，如图 6-16 所示。

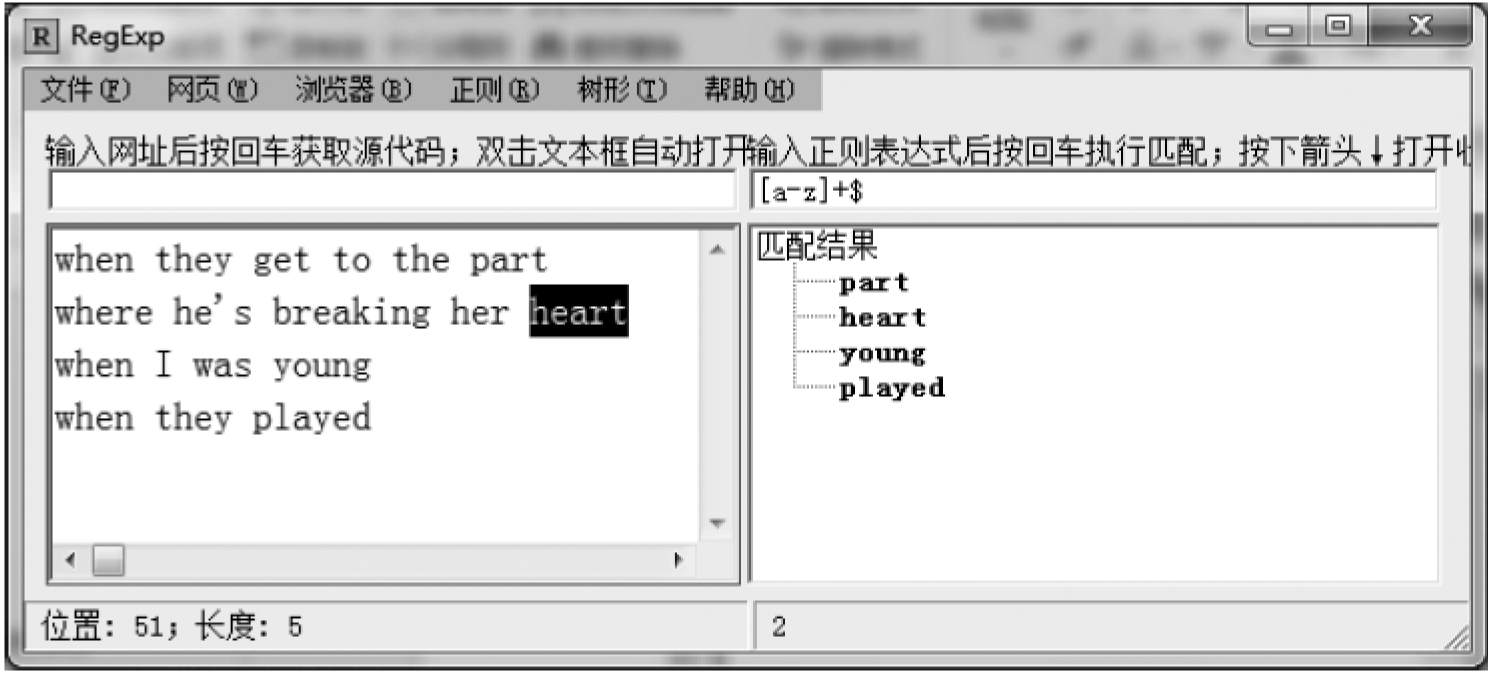


图 6-16　匹配行尾

6.5.4　重复多次

在一个字符或者一个字符范围后面加上花括号，表示前面字符重复多次的意思。

例如 \d{4} 表示恰好匹配 4 个数字，\d{,4} 表示 0 ～ 4 个数字，\d{4,} 表示 4 个以上数字，\d{4,7} 表示 4 ～ 7 个数字。

表达重复个数范围，还有以下 3 种简写形式。

- {0,}：0 个以上，简写为 *。
- {1,}：1 个以上，简写为 +。
- {0,1}：0 个或 1 个，简写为 ?。

例如 \d+\.?\d+ 去匹配下面这行文字：

“小王年龄 36 岁，身高 1.75m。”

可以匹配到 36、1.75。

因为 \d+ 表示 1 个以上的数字，中间的 \.? 表示最多有 1 个小数点。

6.5.5 贪婪和非贪婪

正则表达式中 ? 的含义相当复杂，使用场合不同，含义也随之不同。

? 的含义一般是 {0,1}，表示重复次数。要表达问号本身，需要使用 \?。

例如 Pattern 为 老李吧 \?? 你怎么样，可以匹配老李吧你怎么样，中间的问号可有可无。

此外，? 还经常跟在重复次数的后面，此时的含义是非贪婪，也就是最短匹配。

例如 a{m,n}? 只能匹配到 m 个连续的 a，如果去掉 ?，则尽可能地去匹配 n 个 a。

不使用 ? 时，默认贪婪匹配，也就是尽可能多向后搜索。例如 [A-Za-z]{2,} 表示匹配 2 个以上连在一起的英文字母，如图 6-17 所示。

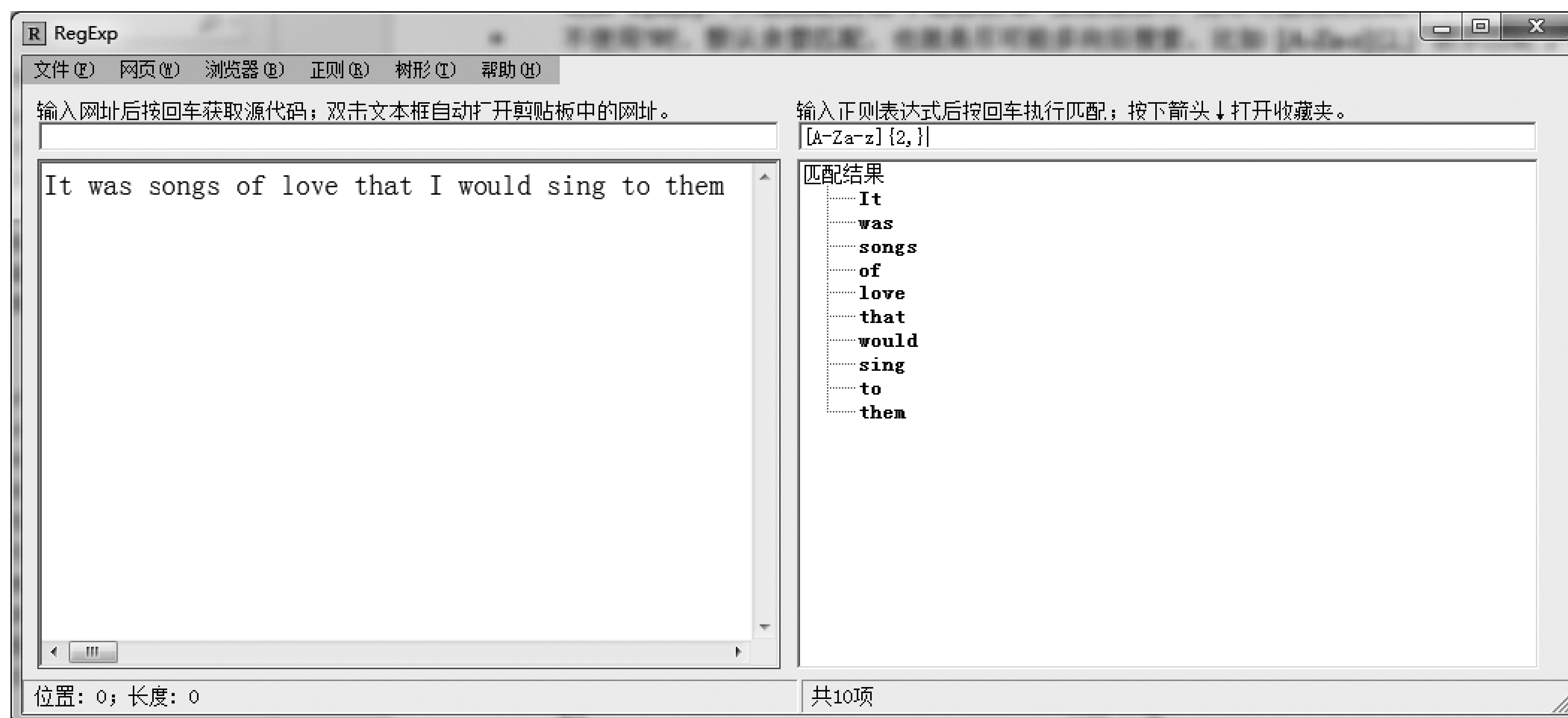


图 6-17 贪婪模式

在花括号后面加 ?，成为非贪婪模式，此时尽可能形成比较短的匹配，只要凑够 2 个连续的英文字母就成为一个匹配内容，如图 6-18 所示。

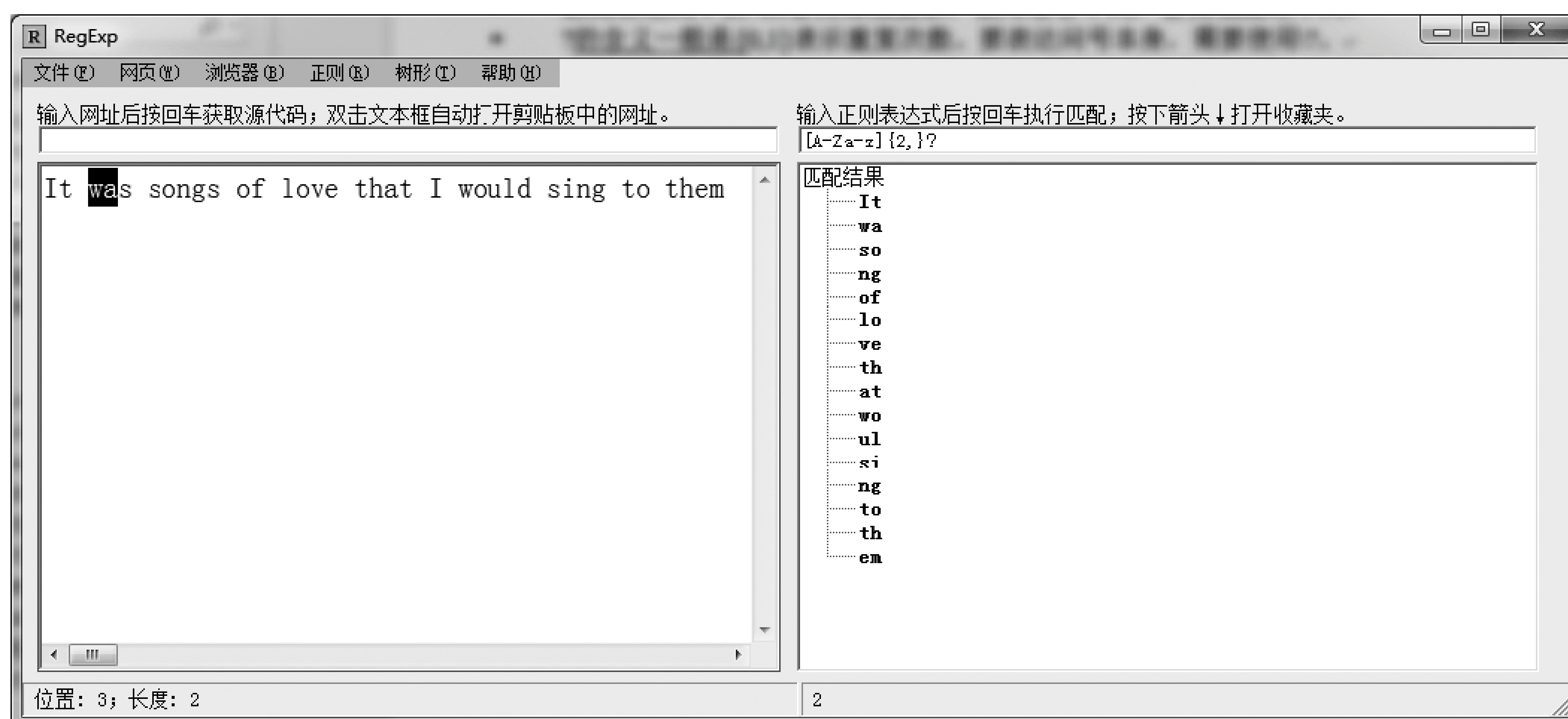


图 6-18 非贪婪模式

6.6　正则表达式测试器

正则表达式测试器是笔者利用 VB6 开发的 Windows 应用程序，基本的功能就是在左侧文本框输入源文本，在右侧输入正则表达式，然后按下回车键查看输出的匹配结果。

新版的正则表达式测试器还增加了获取网页源代码的功能，输入网页的 url 并按下回车键，网页源代码会出现在左侧文本框内成为正则表达式的源文本，如图 6-19 所示。

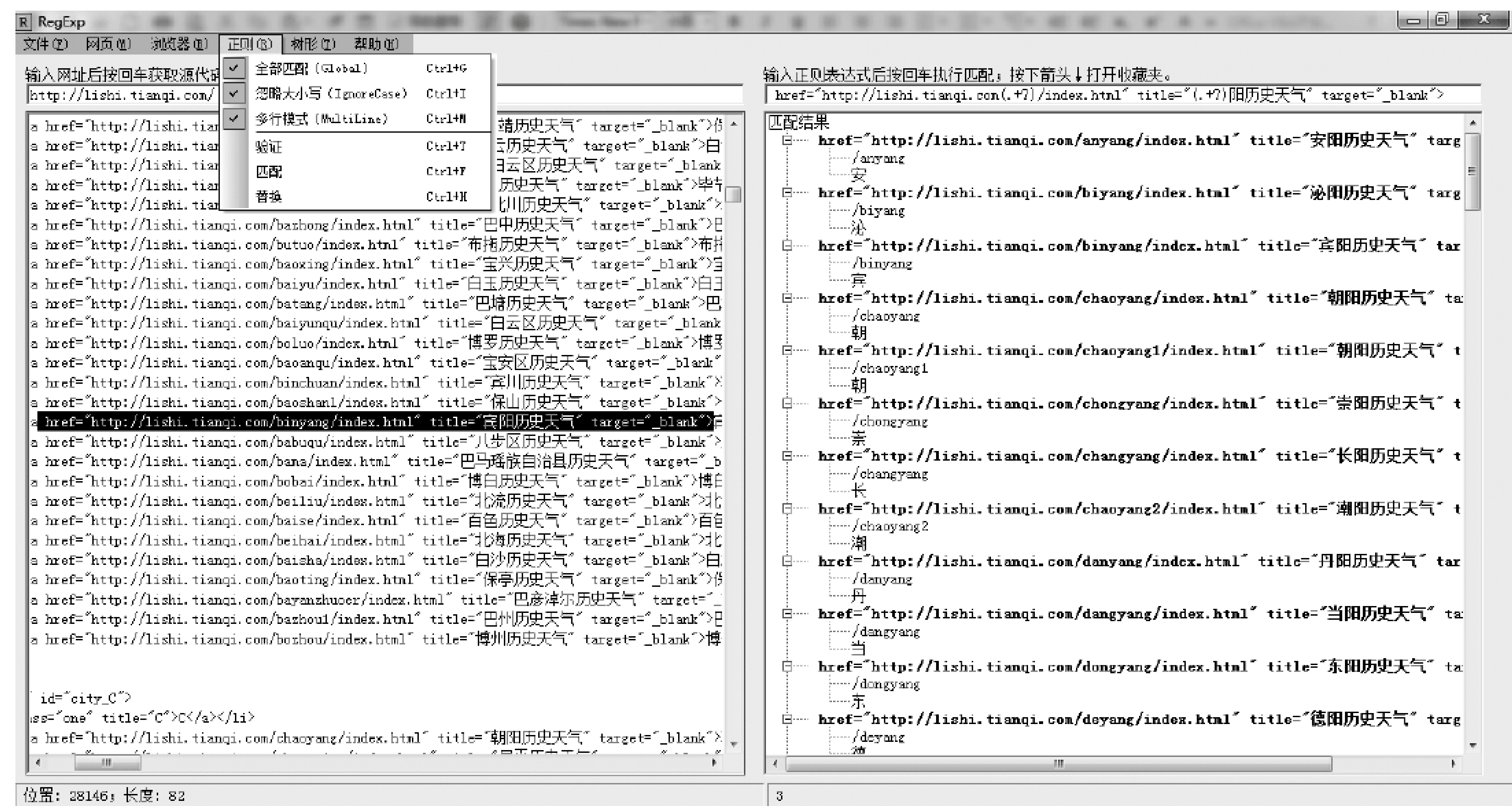


图 6-19　正则表达式测试器界面

软件还支持附件、表格自动下载、模拟真实浏览器的功能，读者可以从本书配套资源下载到本软件。

6.7　本章小结

正则表达式是字符串处理最强大的工具，可以在很多编程语言中使用。

正则表达式最重要的必需属性是 Pattern，这个属性指明了要查找的目标。Test 方法用于验证源字符串是否符合指定的格式，Replace 方法用于把查找到的目标替换为指定的字符串。

正则表达式的 Execute 方法用于查找，通常返回多个查找结果，形成一个 MatchCollection 对象，该对象的个体对象是 Match，如果 Pattern 中包含分组，还可以使用 SubMatches 从 Match 中取出每个分组。

第 7 章 使用字典

字典（Dictionary）是一种键值对（Key-Value pair）的集合对象，通常唯一的键可以直接访问到对应的值。在很多情况下使用字典比数组要方便很多。

字典在形式上特别类似于只有两列的二维数组，其中第 1 列称作“键”（Key），第 2 列称作“值”（Value）。其中键不允许重复，如图 7-1 所示。

键	值
邯郸	0310
石家庄	0311
保定	0312
张家口	0313
承德	0314
唐山	0315
廊坊	0316
沧州	0317
衡水	0318
邢台	0319
秦皇岛	0335

图 7-1　字典的键和值

城市和它对应的区号就形成了一个键值对，例如：保定→0312是一个键值对，保定是键，0312 是值。图中所示的字典总共有 11 个键值对（11 行）。

假如要查询保定的区号是多少，dic(" 保定 ") 就可以返回 0312，无须遍历每个城市。

从总体上讲，一个字典的所有键构成了一个 Keys 对象，该对象表达了字典中的所有键，可以传递给一维数组。字典的所有值构成了一个 Items 对象，也可以传递给一维数组。

本章讲解在 VBA 编程过程中字典的创建、维护，以及使用字典实现去除重复项的具体应用案例。

本章用到的外部引用和重要对象：

- Microsoft Scripting Runtime
  - Scripting.Dictionary

7.1　字典对象

在 VBA 中使用字典，需要添加外部引用“Microsoft Scripting Runtime”，前面介绍 FSO 对象处理文件时曾经使用过这个引用，如图 7-2 所示。

后期绑定创建 Dictionary 对象的方法：CreateObject("Scripting.Dictionary")。

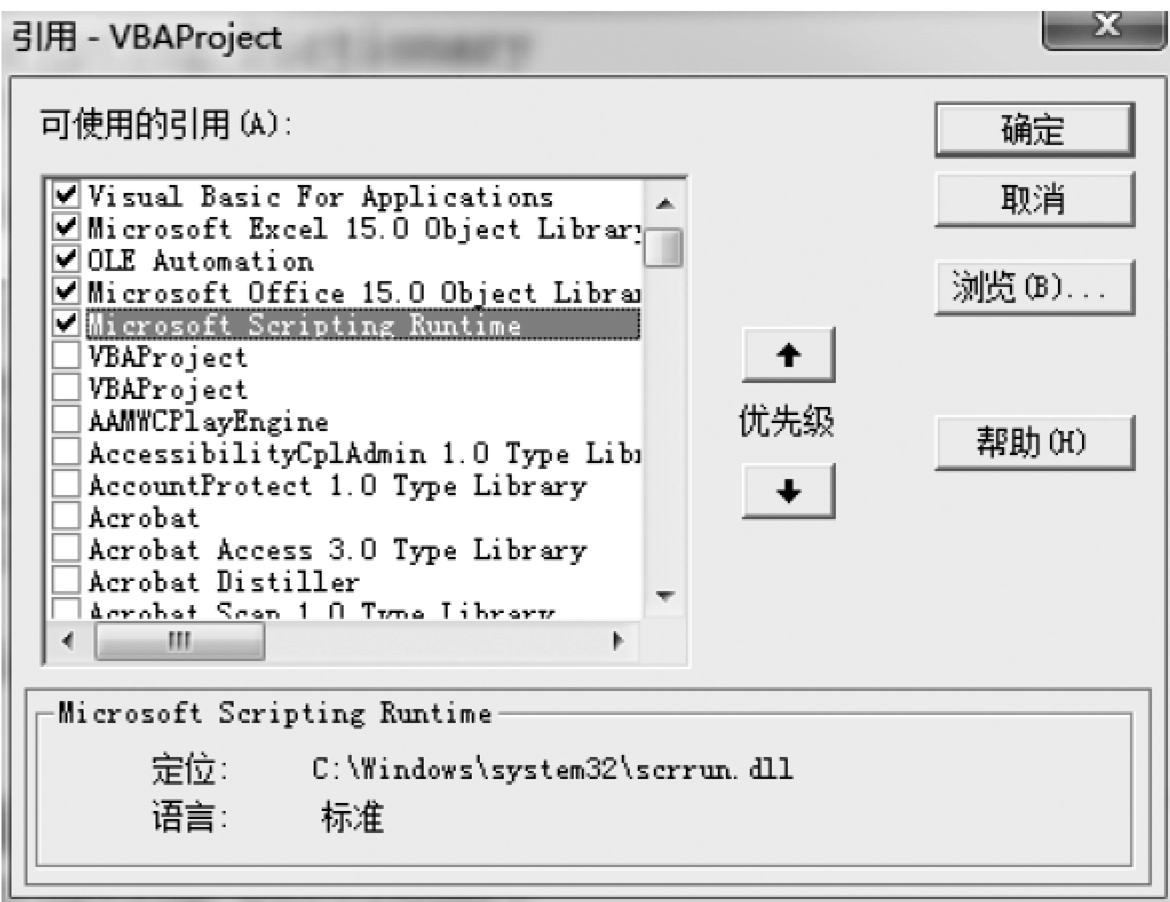


图 7-2　添加外部引用

7.1.1　字典的属性和方法

当 VBA 工程添加了字典的外部引用，按下【 F2 】键打开 VBA 的对象浏览器，在类别中选择 " Scripting"，关键字搜索 " dictionary"，可以看到 Dictionary 对象的所有属性和方法，如图 7-3 所示。

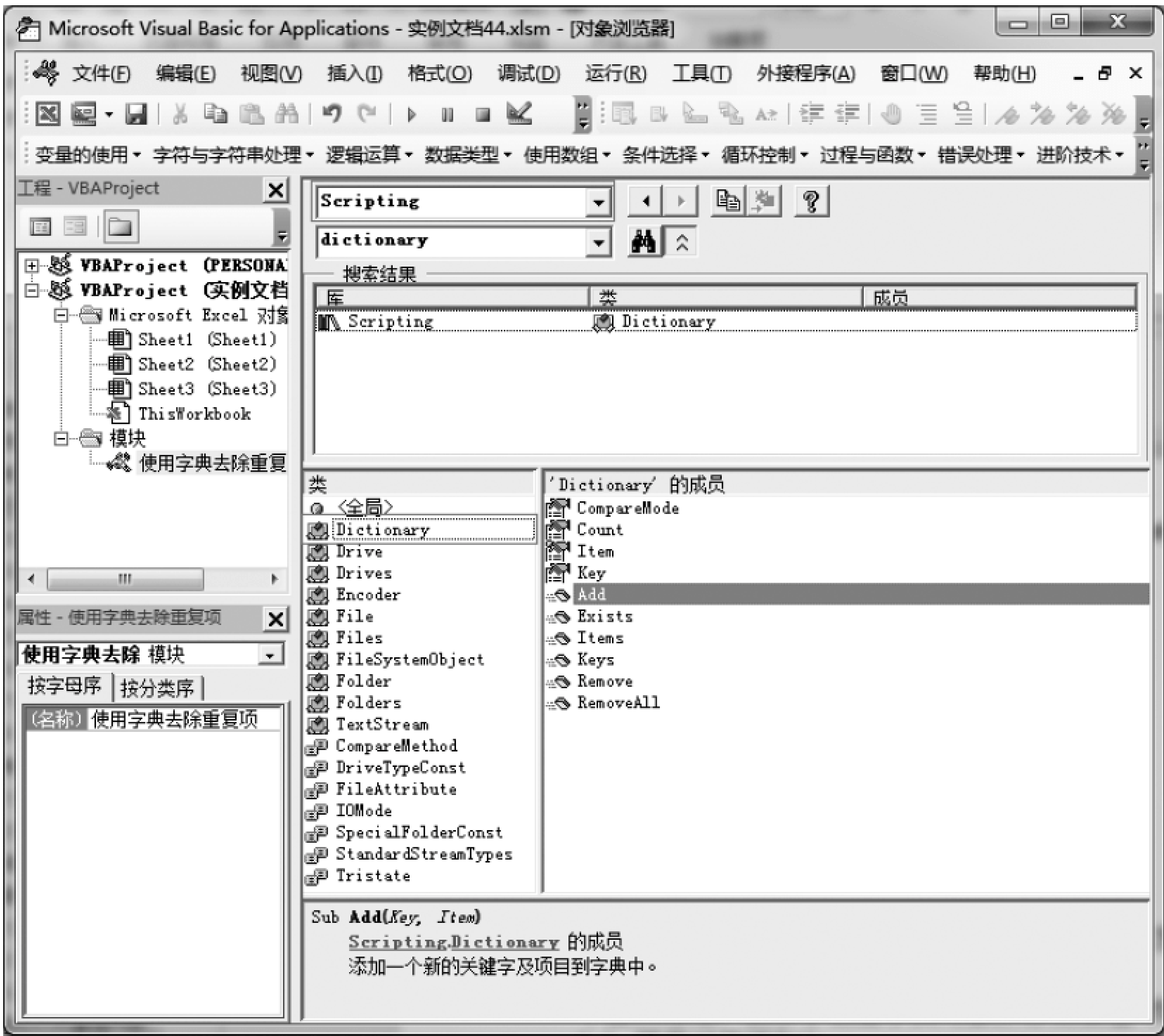


图 7-3　字典对象的成员

了解 Dictionary 对象的成员有助于快速掌握字典的使用技巧，最主要的成员和功能如表 7-1 所示。

表 7-1 Dictionary 对象的主要成员及其功能

成 员 名 称	功 能	代 码 示 例
CompareMode	设置字典的键是否区分大小写，默认为二进制比较模式	dic.CompareMode=TextCompare
Count	返回字典所包含的键值对总数	MsgBox dic.Count
Item	设置或返回字典指定键名的键值	MsgBox dic.Item(" 北京 ")
Key	修改键名	dic.Key(" 北京 ")="Beijing"
Add	添加新的键值对，如果已存在键名，则会出错	dic.Add " 北京 ", 2008
Exists	判断是否存在某键名	MsgBox dic.Exists(" 北京 ")
Items	所有键值组成的数组	arr=dic.Items
Keys	所有键名组成的数组	arr=dic.Keys
Remove	移除指定键名的键值对	dic.Remove " 北京 "
RemoveAll	移除所有键值对	dic.RemoveAll

7.1.2 键值对的添加

创建一个新的字典对象后，就可以使用 dic.Add 方法或者 dic.Item(" 键 ")= 值 的形式添加新的键值对。

下面的过程演示了创建字典对象并添加键值对的过程。

```
Sub 键值对的添加 ()
    Dim dic As Scripting.Dictionary
    Set dic = New Scripting.Dictionary
    With dic
        .Add " 邯郸 ", "0310"
        .Add " 石家庄 ", "0311"
        .Add " 保定 ", "0312"
        .Add " 张家口 ", "0313"
        .Add " 承德 ", "0314"
        .Add " 唐山 ", "0315"
        .Add " 廊坊 ", "0316"
        .Add " 沧州 ", "0317"
        .Add " 衡水 ", "0318"
        .Add " 邢台 ", "0319"
        .Add " 秦皇岛 ", "0335"
        Debug.Print " 键值对总数: ", .Count
        Debug.Print " 廊坊的区号是: ", .Item(" 廊坊 ")
    End With
End Sub
```

代码分析：根据键获取其对应值的表达形式是 dic.Item(" 廊坊 ")，其中 Item 可以不写，因此 dic(" 廊坊 ") 就是廊坊对应的区号。

运行上述过程，立即窗口的结果如图 7-4 所示。

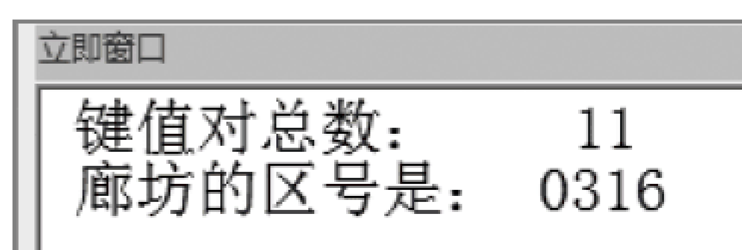


图 7-4 获取键值对的总数以及指定键对应的值

需要注意的是，一个字典对象往往会让不同的模块、过程访问，因此在实际开发中经常把 Dictionary 对象声明为模块级，如果声明为过程级别，当创建字典的代码过程执行完后，字典就自动释放了。

7.1.3　键值对的修改

在实际编程应用中，经常需要在已有键值对的基础上更改个别键值对的内容。

修改值的方法是：dic.Item(" 键 ") = 新值。

修改键名的方法是：dic.Key(" 旧键名 ") = 新键名。

下面的过程为字典添加一部分键值对后修改保定的区号为 312，并且把廊坊的键名修改为 LangFang。

```
Sub 键值对的修改 ()
    Dim dic As Scripting.Dictionary
    Set dic = New Scripting.Dictionary
    With dic
        .Add " 邯郸 ", "0310"
        .Add " 石家庄 ", "0311"
        .Add " 保定 ", "0312"
        .Add " 张家口 ", "0313"
        .Add " 承德 ", "0314"
        .Add " 唐山 ", "0315"
        .Add " 廊坊 ", "0316"
        .Add " 沧州 ", "0317"
        .Add " 衡水 ", "0318"
        .Add " 邢台 ", "0319"
        .Add " 秦皇岛 ", "0335"

        .Item(" 保定 ") = "312"
        .Key(" 廊坊 ") = "LangFang"

        Debug.Print " 保定区号：", .Item(" 保定 ")
        Debug.Print " 廊坊区号：", .Item("LangFang")
    End With
End Sub
```

运行上述代码，立即窗口的结果如图 7-5 所示。

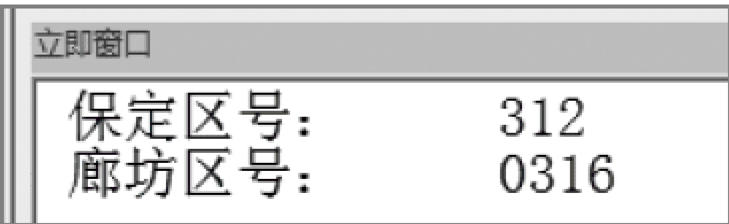


图 7-5　修改键和值

特别提示

使用 dic.Item 方法时，如果键名已经存在，则修改现有键值；如果键名不存在，则新建一个键值对。例如以下 3 行代码。

```
dic.Add " 语文 ", 88
dic.Item(" 数学 ") = 95
dic.Item(" 语文 ") = 79
```

第 1 行代码为字典添加“语文”的成绩为 88 分。由于字典目前不存在“数学”，所以第 2 行的功能是添加“数学”成绩 95 分。在第 3 行代码中，由于字典中已存在“语文”成绩，所以该行相当于刷新语文成绩为 79 分。

因此，执行上述 3 行代码后，dic 总共有 2 个键值对，语文 =79，数学 =95。

7.1.4 键值对的移除

字典对象的 Remove 方法用于移除一个键值对，RemoveAll 方法用于清空字典。

例如 dic.Remove Key:=" 邢台 "，就移除邢台这个键值对，字典的 Count 属性也相应减少 1。

dic.RemoveAll 用于清空整个字典，当然，也可以用 Set dic=New Dictionary 重新创建新字典。

7.1.5 指定的键是否存在

字典的 Exists 方法可以验证某个键是否在字典里面。很多情况下 Exists 方法和 Add 方法配合使用，可以选择性地添加新的键值对。

如果字典中已经存在一个键，那么不能再用 Add 方法继续添加该键名，否则会弹出错误对话框。

```
If dic.Exists(" 辛集 ") Then
   MsgBox " 已存在辛集! "
Else
   dic.Add " 辛集 ", "0311"
End If
```

7.1.6 遍历字典

字典由两列构成，而且键列和值列的行数一样，因此在遍历字典的时候，通常采用键和值分别遍历的方式。

下面的过程使用了 3 个 For 循环，分别遍历字典的键、值、键值对。

```
Sub 键值对的修改 ()
    Dim dic As Scripting.Dictionary
    Dim v
    Dim i As Integer
    Set dic = New Scripting.Dictionary
    With dic
        .Add " 邯郸 ", "0310"
        .Add " 石家庄 ", "0311"
        .Add " 保定 ", "0312"
        .Add " 张家口 ", "0313"
        .Add " 承德 ", "0314"
        .Add " 唐山 ", "0315"
        .Add " 廊坊 ", "0316"
        .Add " 沧州 ", "0317"
        .Add " 衡水 ", "0318"
        .Add " 邢台 ", "0319"
        .Add " 秦皇岛 ", "0335"

        For Each v In .Keys
            Debug.Print v
```

```
            Next v

            For Each v In .Items
                Debug.Print v
            Next v

            For i = 0 To .Count - 1
                Debug.Print .Keys(i), .Items(i)
            Next i
        End With
    End Sub
```

代码分析：当采用数字下标遍历时，要注意字典对象的下界是从 0 开始，因此循环遍历的终止值是 dic.Count−1。

由于字典的所有键 Keys 相当于一个一维字符串数组，因此，可以整体发送到 Excel 单元格中。

下面的过程分别把字典的键、值整体发送到相应单元格区域中。

```
    Sub 发送到单元格 ()
        Range("A1").Resize(, dic.Count) = dic.Keys
        Range("A2").Resize(, dic.Count) = dic.Items
        Range("B5").Resize(dic.Count) = Application.WorksheetFunction.Transpose
(dic.Keys)
        Range("C5").Resize(dic.Count) = Application.WorksheetFunction.Transpose
(dic.Items)
    End Sub
```

代码分析：如果发送到一行中，需要把单元格区域的列数用 Resize 方法扩展到和字典键值对总数一样才行。

如果发送到一列中，要扩展单元格区域的行数，并且采用工作表的转置函数处理一下。

运行上述过程，单元格中显示字典的所有键值对，如图 7-6 所示。

	A	B	C	D	E	F	G	H	I	J	K	L
1	邯郸	石家庄	保定	张家口	承德	唐山	廊坊	沧州	衡水	邢台	秦皇岛	
2	310	311	312	313	314	315	316	317	318	319	335	
3												
4												
5		邯郸	310									
6		石家庄	311									
7		保定	312									
8		张家口	313									
9		承德	314									
10		唐山	315									
11		廊坊	316									
12		沧州	317									
13		衡水	318									
14		邢台	319									
15		秦皇岛	335									
16												

图 7-6　把字典的键和值发送到横向、纵向单元格区域

7.1.7　字典的比较模式

字典的 CompareMode 属性有以下 3 种取值。

- BinaryCompare：二进制模式。

❑ DatabaseCompare：数据库模式。

❑ TextCompare：文本模式。

默认属性是 BinaryCompare，也就是严格区分大小写，这种情况下，如果字典中已经有 China 这个键，那么再添加一个 china 也是可以的。

如果创建字典并设置比较模式为 TextCompare，那么会把 China 和 china 看作是相同的键，如图 7-7 所示。

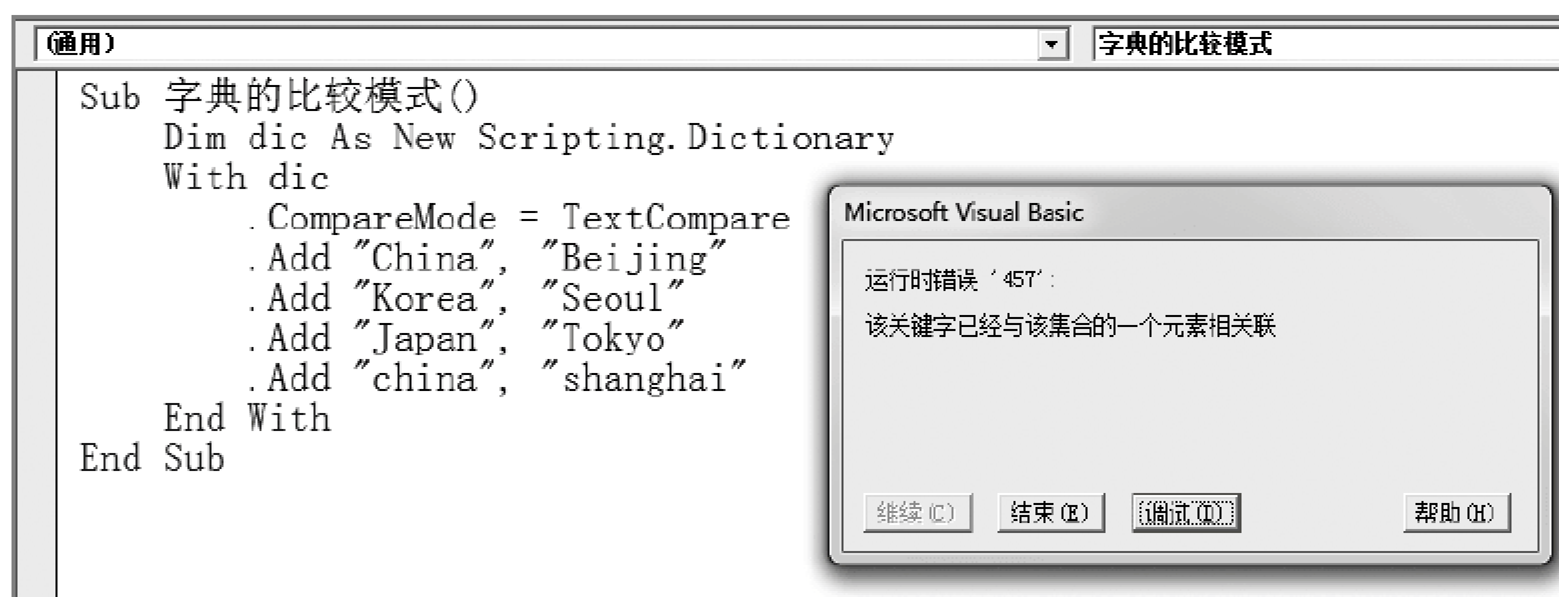


图 7-7　文本模式下大写和小写的键名被认为相同

7.1.8　字典的数据类型

为了讲解方便，前面所述实例中的键和值均采用了字符串类型，实际上字典的键、值还可以是其他数据类型。

下面过程中用到了 4 个字典，每个字典的键和值用了不同的数据类型。

```
Sub 不同数据类型 ()
    Dim D1 As New Scripting.Dictionary
    Dim D2 As New Scripting.Dictionary
    Dim D3 As New Scripting.Dictionary
    Dim D4 As New Scripting.Dictionary
    With D1
        .Add 1, #1/31/2018#
        .Add 2, #2/28/2018#
        .Add 3, #3/31/2018#
        .Add 4, #4/30/2018#
        Debug.Print .Item(3)            '查询键名为 3 的值
    End With

    With D2
        .Add True, "Yes"
        .Add False, "No"
        Debug.Print .Item(False)
    End With

    With D3
        .Add 3.14, 22 / 7
        .Add 2.71828, VBA.Exp(1)
```

```
        Debug.Print .Item(3.14)
    End With

    With D4
        .Add Key:="App", Item:=Application
        .Add Key:="Wbk", Item:=Application.ActiveWorkbook
        .Add Key:="Wst", Item:=Application.ActiveSheet
        .Add Key:="Rng", Item:=Application.ActiveCell
        Debug.Print .Item("App").UserName
        Debug.Print .Item("Wbk").Name
        Debug.Print .Item("Wst").Name
        Debug.Print .Item("Rng").Address
    End With
End Sub
```

代码分析：上述程序包含 4 个不同的字典，D1 的数据类型是 Integer → Date，D2 是 Boolean → String，D3 是 Double → Double，D4 是 String →对象变量。

运行上述过程，立即窗口的结果如图 7-8 所示。

以上内容的源代码文件为“实例文档 43.xlsm”。

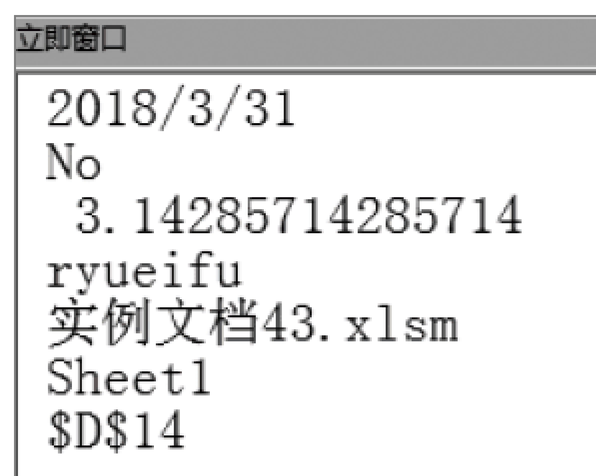


图 7-8　运行结果

7.2　字典的应用

字典的特性是不能有重复的键名。因此经常利用这个特性实现去除重复。

7.2.1　提取单列数据中的唯一值

假设工作表 A 列中有一些姓名，很多姓名出现了多次，现在要求不重复的姓名有哪些。

下面的代码把姓名作为字典的键名创建键值对，至于每个键对应的值，什么都可以，本例直接赋空字符串。

```
Sub 单列去重()
    Dim dic As New Scripting.Dictionary
    Dim i As Integer
    With dic
        For i = 1 To 15
            .Item(Range("A" & i).Value) = ""
        Next i
        Range("C1").Resize(.Count).Value = Application.WorksheetFunction.Transpose
(.Keys)
    End With
End Sub
```

代码分析：For 循环体结束后，字典的总数一定比循环次数少，最后把字典的所有键发送到单元格。结果显示：不重复的姓名仅仅 5 个，如图 7-9 所示。

C5 李四

	A	B	C	D	E	F
1	赵六		赵六			
2	王五		王五			
3	张三		张三			
4	张三		小明			
5	小明		李四			
6	李四					
7	张三					
8	王五					
9	王五					
10	张三					
11	赵六					
12	赵六					
13	赵六					
14	赵六					
15	李四					
16						

图 7-9　利用字典去除重复项

7.2.2　删除重复行

虽然 Excel 2010 以上版本都自带删除重复项的功能，这里为了演示字典的特性，借助字典标记和删除重复行。

假设工作表中有一个学生成绩表，下面的代码可以把重复的记录底纹颜色设置为红色。然后再把所有底纹为红色的单元格删除。

```
Sub 删除重复行()
    Dim dic As New Scripting.Dictionary
    Dim i As Integer
    With dic
        For i = 2 To 15
            If .Exists(Range("A" & i).Value) Then
                Range("A" & i).Resize(, 4).Interior.Color = vbRed
            Else
                .Add Key:=Range("A" & i).Value, Item:=""
            End If
        Next i
        MsgBox "接下来删除重复行！", vbInformation
        For i = 15 To 2 Step -1
            If Range("A" & i).Resize(, 4).Interior.Color = vbRed Then
                  Range("A" & i).Resize(, 4).Delete Shift:=Excel.XlDeleteShift
Direction.xlShiftUp
            End If
        Next i
    End With
End Sub
```

代码分析：本过程包含 2 个循环结构，第一个循环用来把 A 列的姓名添加到字典，如果字典中已经存在某个姓名，则认为是重复记录，就把该行设置为红色。

第一个循环体结束后，所有重复记录都加上了红色，如图 7-10 所示。

为了删除这些加了颜色的单元格，需要用倒序删除，只要是红色的单元格区域就删除。

最后结果如图 7-11 所示。

	A	B	C	D	E
1	姓名	语文	数学	英语	
2	赵六	81	90	82	
3	李四	80	98	61	
4	大龙	70	64	67	
5	王五	78	98	82	
6	王五	78	98	82	
7	小虎	98	87	78	
8	王五	78	98	82	
9	王五	78	98	82	
10	大龙	70	64	67	
11	小虎	98	87	78	
12	大龙	70	64	67	
13	赵六	81	90	82	
14	王五	78	98	82	
15	小虎	98	87	78	
16					
17					

图 7-10　重复记录自动标记为红色

D12

	A	B	C	D
1	姓名	语文	数学	英语
2	赵六	81	90	82
3	李四	80	98	61
4	大龙	70	64	67
5	王五	78	98	82
6	小虎	98	87	78
7				

图 7-11　删除所有填充色为红色的行

7.2.3　检查字符串中是否有重复字符

某些场合需要判断一个字符串中的每个字符是不是唯一的，是否有出现两次以上的情况。例如“Condition”这个单词中，i 和 o 都出现了多次，而“Friend”这个单词无重复情况。下面的过程把字符串中的每个字符作为字典的键添加到字典中，如果有重复字符，字典的键值对个数一定小于字符串的长度，利用这个特征进行判断。

```
Sub 检查是否有重复字符 ()
    Const s As String = "Microsoft"
    Dim dic As New Dictionary
    Dim i As Integer
    For i = 1 To Len(s)
        dic.Item(Mid(s, i, 1)) = ""
    Next i
    If dic.Count < Len(s) Then
        MsgBox "有重复字符"
    Else
        MsgBox "无重复字符"
    End If
End Sub
```

运行以上过程，对话框显示“有重复字符”。

以上内容的源代码文件为“实例文档 44.xlsm”。

7.3　本章小结

字典的结构类似于一个多行、2 列的二维数组，由于第一列是不重复的，因此可以方便地通过第一列访问到第二列，不使用循环结构就能通过键找到对应的值。

字典对象的 Exists 方法可以判断字典中是否存在某个键，这些都是数组不具备的性能。

第 8 章 操作数据库

数据库（Database）是存储数据的容器。对于大中型应用程序和软件的开发，一般都需要利用数据库来保存数据、维护数据和读出数据，如果程序中用到的数据量不大，可以考虑利用系统的注册表作为数据容器，也可以考虑使用外部文本文件，但如果软件中有大量的数据进出，使用数据库更为专业。

目前数据库有很多种格式，常见的主要有微软公司的 Microsoft SQL Server、Access 等，另外 Excel 文件、CSV 文件和文本文件也可以用作数据库。

数据库在使用方面，一般分为手工操作和使用代码访问两种方式。本章首先简单演示一下手工在 Access 2013 中创建和维护数据库的方法，然后介绍使用 ADO 技术对数据库进行 SQL 查询。

利用代码访问数据库的方式有很多种，但是 ADO（ActiveX Data Objects）是微软公司推出的一种数据访问技术，优点是易于使用、速度快、占用磁盘空间小。而 SQL（Structured Query Language）是访问和处理数据库的标准语言，仅仅使用一个文本字符串就可以对数据库、表进行复杂的访问和修改操作。

本章用到的外部引用和重要对象：

- Microsoft ActiveX Data Objects 2.8 Library
  - ADODB.Connection
  - ADODB.Recordset

8.1　Access 数据库概述

Access 数据库是存储于磁盘中的一个扩展名为 .accdb 或 .mdb（Access 2003）的文件。要用 VBA+ADO 技术操作和访问数据库，需要事先创建数据库作为被操作的原料，因此下面介绍用 Access 2013 手工创建数据库的方法。

启动 Access 2013 后，依次单击【文件 / 新建 / 空白桌面数据库】。

在弹出的路径输入对话框中，单击右侧的“打开”按钮，可以弹出“文件新建数据库”对话框，改变到自己常用的文件夹，并且输入数据库的名称 ChinaProvince.accdb，如图 8-1 所示。该数据库用来存储中国各省份信息。



图 8-1　创建新数据库

单击“确定”按钮，并且单击“创建”按钮，即可在磁盘文件夹中看到该数据库，如图 8-2 所示。

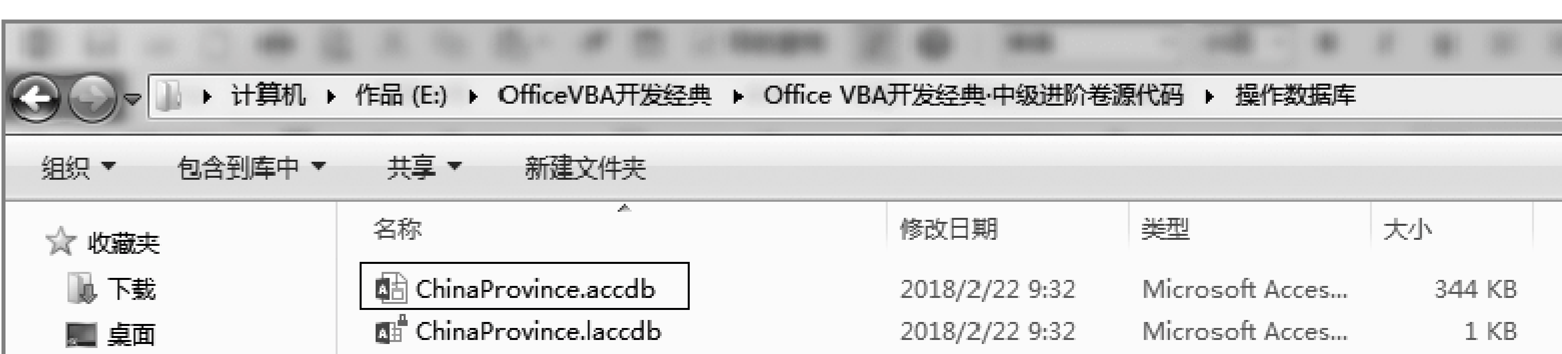


图 8-2　数据库文件和临时文件

需要注意的是，当数据库处于打开状态，或者正在被程序访问，会在数据库文件夹中产生一个扩展名为 .laccdb 的临时文件，这个文件一旦产生，就表示该数据库不可以进行重命名或删除操作，因为数据库处于打开状态。

对于新创建的空白数据库，其中没有任何数据表对象。

一个数据库由查询、表、窗体等多种对象构成，但是表（Table）是最常用的数据存储单位，表从结构上非常类似于 Excel 中的二维表格数据。

一个数据库可以包含 1 个以上的表，每个表有一个名称来唯一识别。

8.1.1　数据表设计

重新打开 ChinaProvince 数据库，单击功能区的【创建 / 表格 / 表设计】，弹出一个“表1”的设计窗格，如图 8-3 所示。

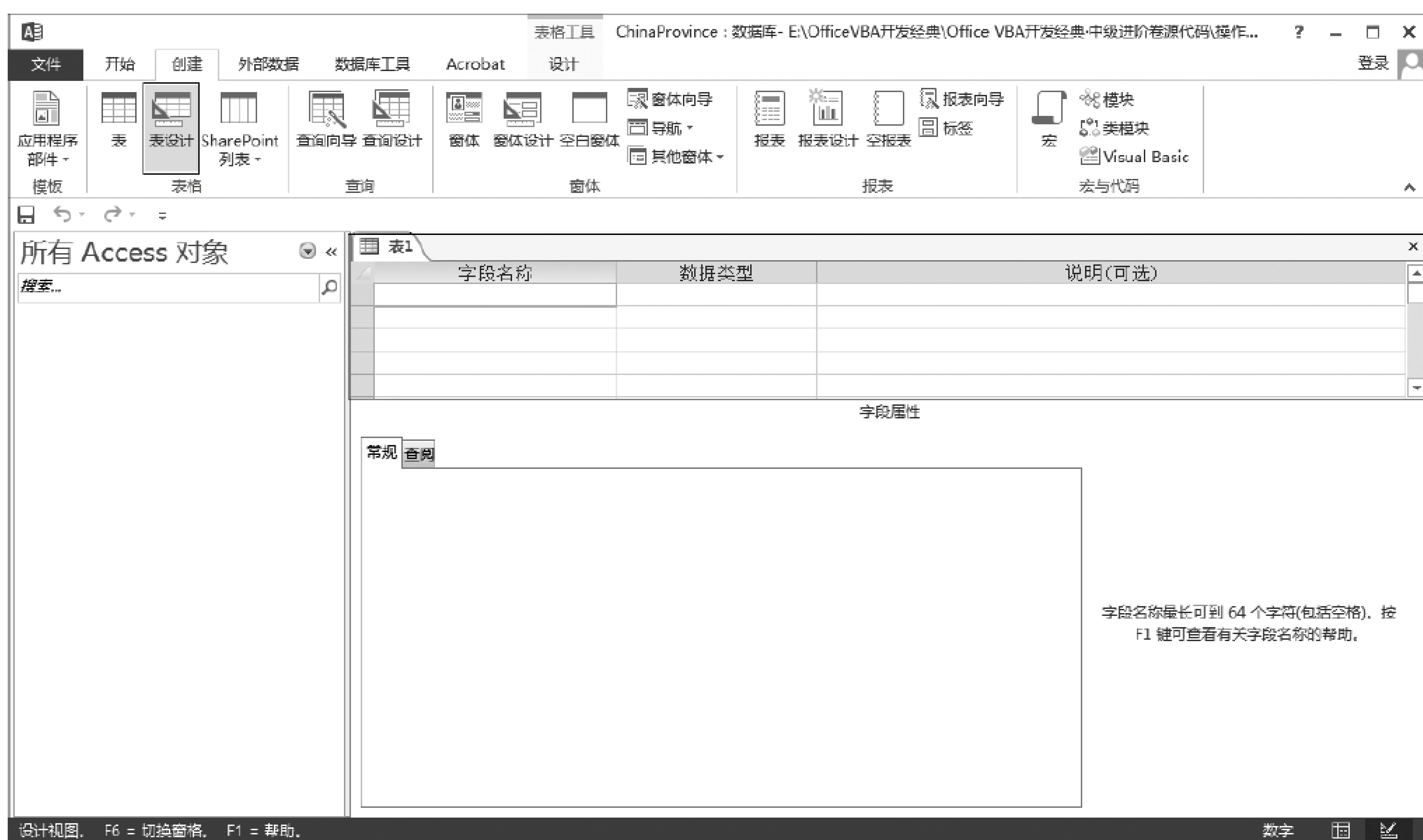


图 8-3　表的设计视图

实际上，“表 1”是默认的命名方式，当没有保存时，该表还没有真正存储在数据库中，因此接下来输入该表的所需字段名称（Field Name）和字段类型（Field Type）。

默认的字段类型都是文本，但是本实例中的面积、人口字段是数字，如果要详细设定数字格式，可以单击下面的窗格。

另外，还有一个少数民族自治区字段，它是一个布尔类型。

字段设定好后，单击右上角的 × 关闭设计视图，此时会弹出一个“另存为”对话框，如图 8-4 所示。

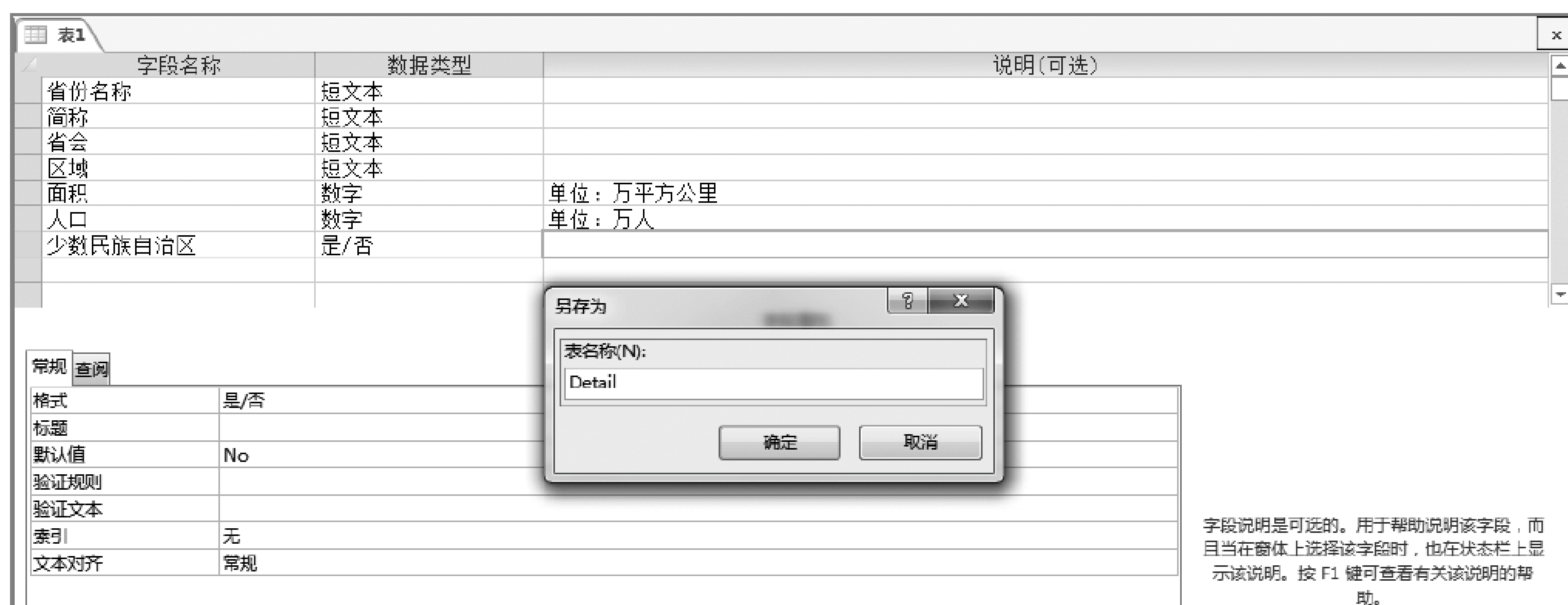


图 8-4　保存表

在“另存为”对话框中输入表的名称：Detail，单击“确定”按钮，此时该表才真正保存于数据库中。

表的结构设计好了，但是真正的数据还没有输入，在左侧“所有 Access 对象”窗格中右击 Detail 表，在弹出菜单中选择“打开”命令，就可以像在 Excel 单元格中一样输入记录（Record）了，如图 8-5 所示。

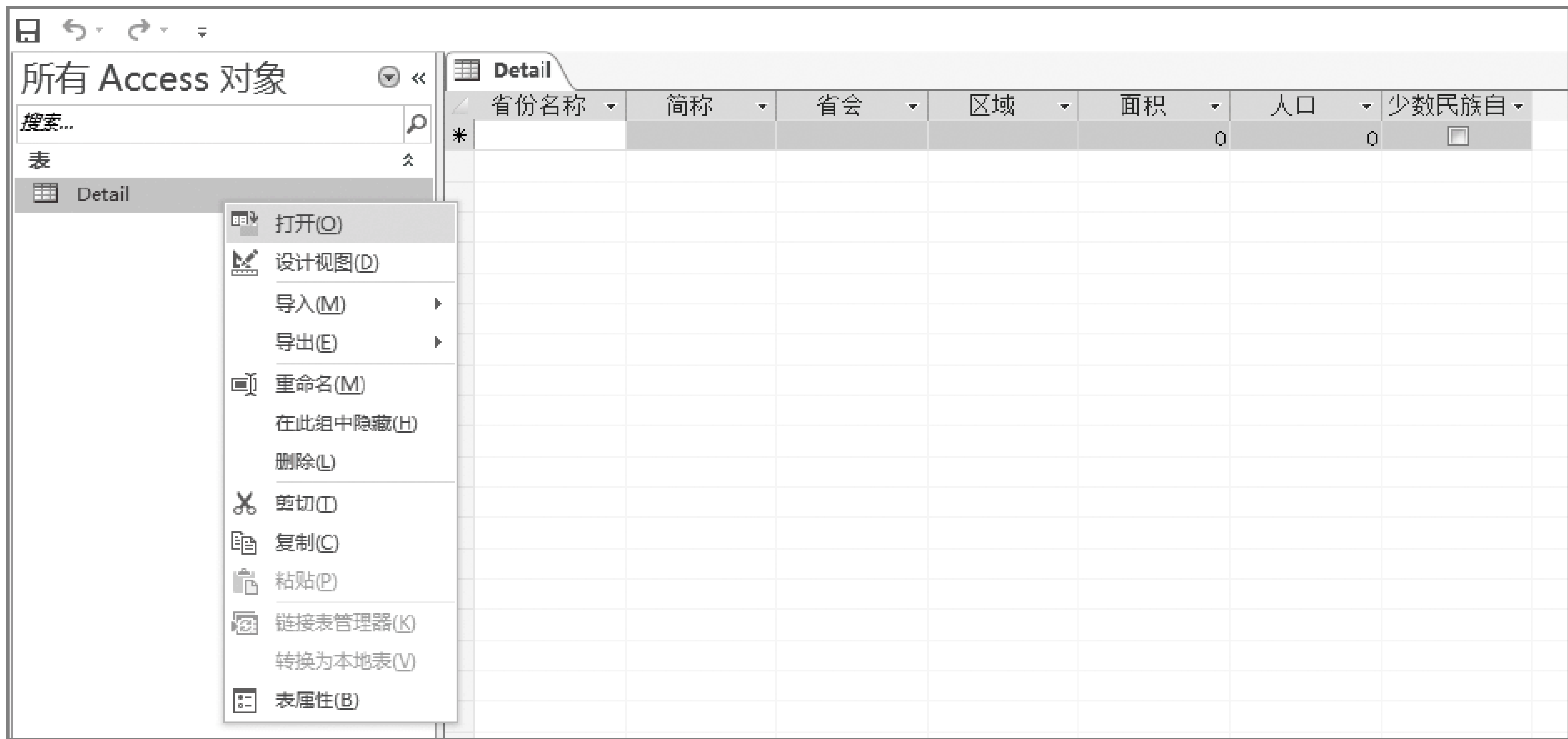


图 8-5　打开表

提示　如果单击右键菜单中的“设计视图”命令，则可以再次对表结构进行调整。

8.1.2　字段类型

Access 常用字段类型和存储数据的种类如下。

- 文本：简短文本。
- 数字：一般数字。
- 日期 / 时间。
- 货币。
- 自动编号：从 1 开始的自动编号，类似于 Excel 中的行号，唯一不重复。
- 是 / 否：布尔型。
- 备注：用于存储比较长的、多行文本等。

一定要记住：Access 数据表和 Excel 工作表不一样，一个字段就是一个列，该列输入的数据必须和规定的数据类型匹配，例如在人口列中输入“三千二百”，就会被拒绝，类似于 Excel 中的有效性验证。

8.1.3　记录维护

数据表的打开视图中显示了当前表中所有的记录，如果某个数据不确定是多少，可以保持空白（Null）。例如，河北的面积、安徽的人口、上海的简称和区域、江西的省会，这 5 个单元格尚未填写，如图 8-6 所示。

数据表左下角有个导航工具，显示表中有 34 个省、自治区或直辖市的信息，当前所选为第 6 项，单击旁边的几个箭头按钮，可以快速定位到前一条、后一条、第一条和最末一条，以及新建记录。

Detail

省份名称	简称	省会	区域	面积	人口	少数民族自
黑龙江	黑	哈尔滨	东北	45.48	3813	☐
吉林	吉	长春	东北	18.74	2699	☐
辽宁	辽	沈阳	东北	15.59	4203	☐
北京	京	北京	华北	1.68	1423	☐
天津	津	天津	华北	1.13	1007	☐
河北	冀	石家庄	华北		6735	☐
山西	晋	太原	华北	15.63	3294	☐
内蒙古	蒙	呼和浩特	华北	118.3	2379	☑
山东	鲁	济南	华东	15.38	9082	☐
江苏	苏	南京	华东	10.26	7381	☐
安徽	皖	合肥	华东	13.97		☐
上海		上海		.63	1625	☐
浙江	浙	杭州	华东	10.2	4647	☐
江西	赣		华东	16.7	4222	☐
福建	闽	福州	华东	12.13	3466	☐
河南	豫	郑州	华中	16.7	9768	☐
湖北	鄂	武汉	华中	18.59	5988	☐
湖南	湘	长沙	华中	21.18	6629	☐
广东	粤	广州	华南	18	7859	☐
广西	桂	南宁	华南	23.6	4822	☑
海南省	琼	海口	华南	3.4	803	☐
香港	港	香港	华南	.11	686	☐
澳门	澳	澳门	华南	.025	44	☐
台湾	台	台湾	华南	3.6	227	☐
四川	川、蜀	成都	西南	48.14	8673	☐
云南	云、滇	昆明	西南	38.33	4333	☐
贵州	贵、黔	贵阳	西南	17.6	3837	☐
重庆	渝	重庆	西南	8.23	3107	☐

记录: 第 6 项(共 34 项) 无筛选器 搜索

图 8-6 数据记录中允许空白

此次就创建好了一个表，根据业务需要还可以向数据库中增加多个表。同时，Access 与 Excel 兼容性非常好，可以很方便地把 Excel 中的数据表批量导入 Access 数据库中，也可以把 Access 数据表导出到 Excel 工作表中。

8.2 使用 ADO 对象操作数据库

ADO 是系统中用来让其他程序、语言访问数据库的一个对象库，本身不属于 VBA 中的对象。因此需要向 VBA 工程中添加外部引用“ Microsoft ActiveX Data Objects 2.8 Library”，如图 8-7 所示。

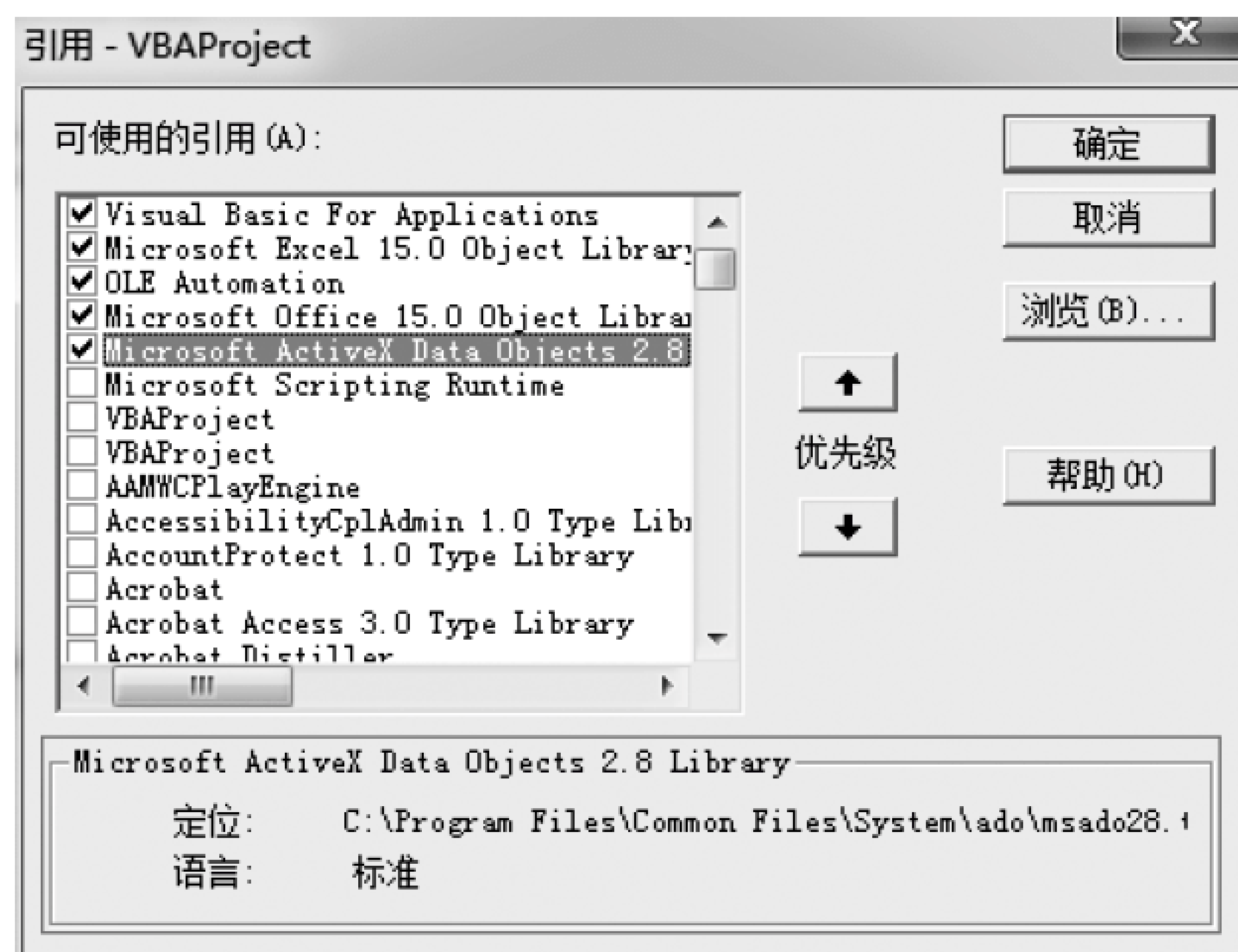


图 8-7 添加外部引用

添加引用后，在对象浏览器中依次输入 ADODB、Connection 并按回车键，可以看到 ADODB 对象库中包含的所有成员，如图 8-8 所示。

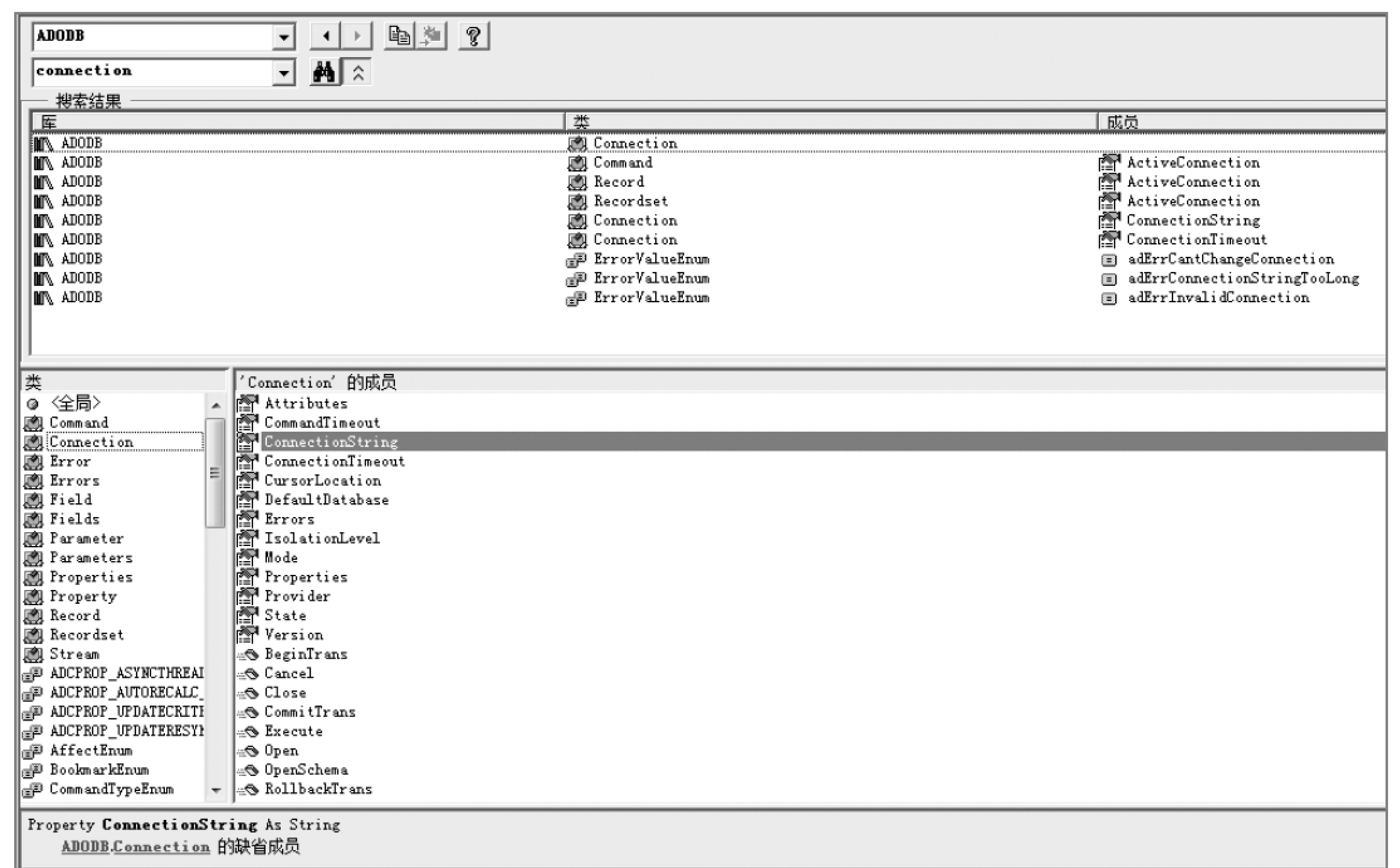


图 8-8　ADODB 对象库的成员

编程过程中比较常用的成员如下。

- ❑ 数据库连接对象 Connection。
- ❑ 记录结果集 RecordSet。
- ❑ 命令执行对象 Command。
- ❑ 字段对象 Field。
- ❑ 属性对象 Property。

其中 Field 是 RecordSet 的成员对象，表示记录集中的字段。Property 是 Connection 的成员对象，表示数据库连接对象的属性。

使用 ADO 查询数据库中表的数据的一般流程如图 8-9 所示。

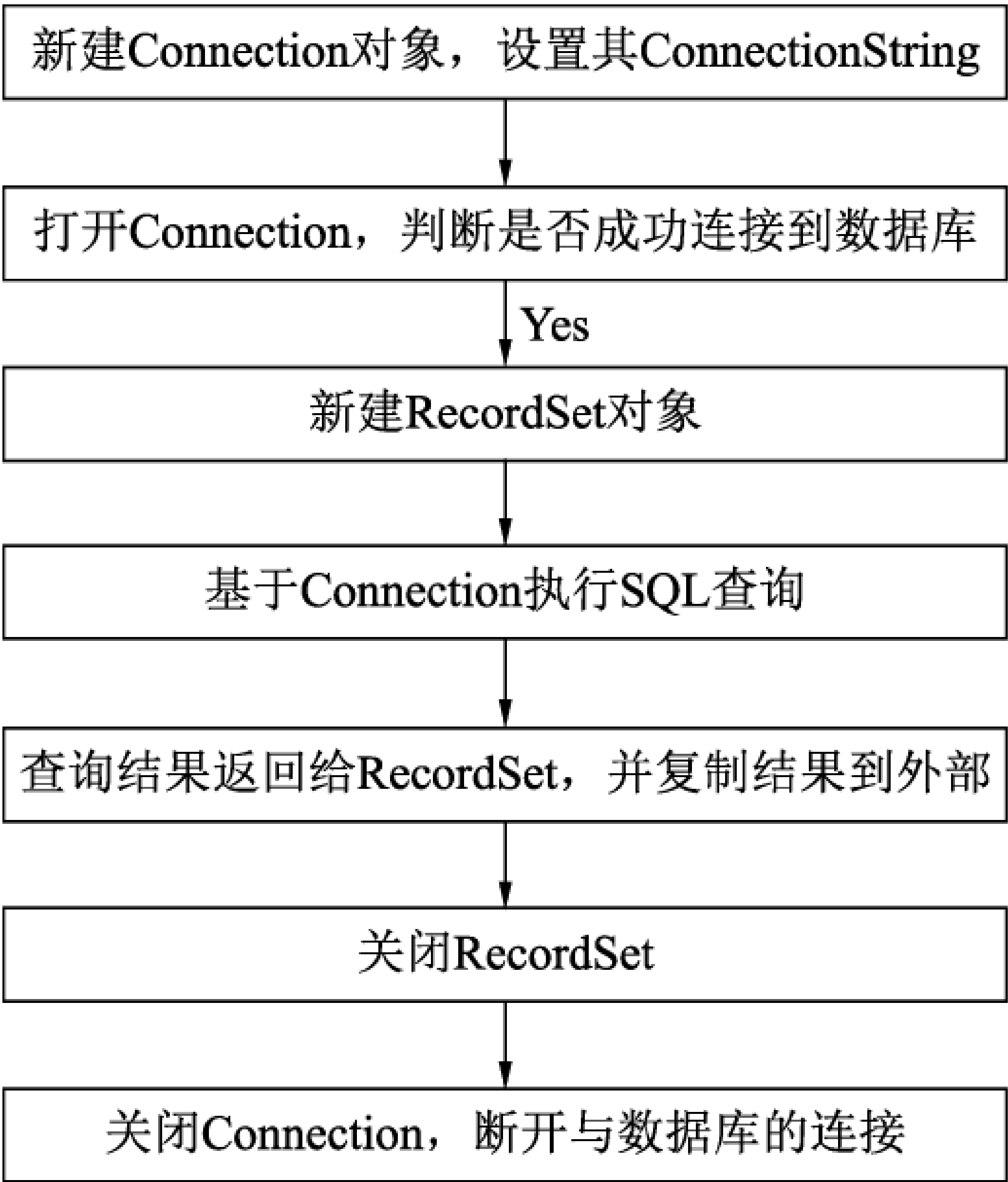


图 8-9　数据库查询的一般流程

8.2.1　Connection 对象

Connection 对象是 ADO 连接并访问数据库的桥梁。它的 ConnectionString 属性是一个文本字符串，用来规定它如何连接到数据库；它的 State 属性可以判断是否连接成功；它的 Properties 成员可以列举其连接数据库的各项属性。

Connection 对象的 Open 方法用来连接数据库，Close 方法关闭数据库的连接，Execute

方法用来执行一个 SQL 查询并返回 RecordSet。

下面的 VBA 过程使用 ADO 连接与工作簿在同一路径下的 ChinaProvince 数据库。

```
    Public cnn As ADODB.Connection, rst As ADODB.Recordset, pro As ADODB.Property,
fld As ADODB.Field
    Sub 连接Access数据库()
        Set cnn = New ADODB.Connection
        With cnn
            .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
            For Each pro In .Properties
                Debug.Print pro.Name, pro.Value '遍历Connection对象的各个属性名和值
            Next pro
            .Open
            If .State = ADODB.ObjectStateEnum.adStateOpen Then
                MsgBox "成功连接到数据库! ", vbInformation
                .Close '只有处于打开的Connection,才能执行Close
            Else
                MsgBox "失败! ", vbExclamation
            End If
        End With
    End Sub
```

代码分析：连接数据库成功与否主要取决于 ConnectionString 的构造，该连接字符串的语法构成如下。

```
    "属性1=值1;属性2=值2;"
```

Provider 规定了连接对象的提供者是 Microsoft.ACE.OLEDB.12.0，对于连接 Access 2003 数据库或者其他种类的数据库，此处需要修改。

Data Source 规定了数据库在磁盘中的位置。

Persist Security Info 表示连接成功后是否保存安全信息，默认为 False。

运行上述过程后，正常情况下会提示“成功连接到数据库！”，如图 8-10 所示。

需要注意的是，无论是 Connection 对象，还是后面要讲的 RecordSet 对象，只有对象处于 Open 状态才能执行 Close 方法。同样，只有对象处于 Close 状态，才能再次 Open 对象。

如果要执行 SQL 查询，就是在 Connection 处于 Open 状态，Close 方法执行之前进行。

如果 ConnectionString 的构造有问题，则会在执行 cnn.Open 那句时出错，如图 8-11 所示。



图 8-10　连接到数据库



图 8-11　连接失败

此时，应该根据错误提示找到出错原因，修改代码后重新运行。

8.2.2　RecordSet 对象

RecordSet 是执行 SQL 查询后形成的结果记录集，也就是过滤筛选后生成的子数据表。

例如从 ChinaProvince 数据库的 Detail 表中找出中国 5 个少数民族自治区的省份名称、省会、区域信息，就需要构造如下 SQL 查询语句。

```
Select 省份名称,省会,区域 From Detail Where 少数民族自治区=True
```

已打开的 Connection 对象执行该查询语句后就形成了一个子表，如图 8-12 所示。

以上这个结果表的信息，通过 RecordSet 对象来访问。

下面的代码在上一个 VBA 程序基础上修改，具体实施查询过程。

	省份名称	省会	区域
▶	内蒙古	呼和浩特	华北
	广西	南宁	华南
	西藏	拉萨	西南
	新疆	乌鲁木齐	西北
	宁夏	银川	西北
*			

图 8-12　SQL 查询产生的结果记录集

```
    Public cnn As ADODB.Connection, rst As ADODB.
Recordset, pro As ADODB.Property, fld As ADODB.Field
    Sub 查询Access数据库()
        Set cnn = New ADODB.Connection
        With cnn
            .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
            .Open
            If .State = ADODB.ObjectStateEnum.adStateOpen Then
                Set rst = New ADODB.Recordset
                rst.CursorLocation = adUseClient
                rst.Open Source:="Select 省份名称,省会,区域 From Detail Where 少数民
族自治区=True", ActiveConnection:=cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset,
LockType:=ADODB.LockTypeEnum.adLockOptimistic
                If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                    Debug.Print "结果记录集的记录总数：", rst.RecordCount
                    ActiveSheet.Range("A2").CopyFromRecordset rst
                    rst.Close '关闭rst
                End If
                .Close '关闭cnn
            Else
                MsgBox "失败！", vbExclamation
            End If
        End With
        Set rst = Nothing
        Set cnn = Nothing
    End Sub
```

代码分析：执行 SQL 查询的核心代码是 rst.Open 那一句，RecordSet 的 Open 方法中通常包含 4 个参数，其中 Source 参数需要规定一个 SQL 查询语句，ActiveConnection 需要设置为当前数据库连接对象，后面两个参数分别为游标类型和锁类型。

ActiveSheet.Range("A2").CopyFromRecordset rst 用来把生成的结果记录集发送到以 A2

单元格为左上角的单元格区域中，运行上述过程，立即窗口打印出记录集的总数为 5，记录集导出到 Excel单元格中，如图 8-13 所示。

Excel VBA 中 Range 对象的 CopyFromRecordset 方法专门用来接收数据库的结果记录集，但是该方法不会自动加上字段标题，这还需要遍历 RecordSet 的 Fields 属性方可。

F10

	A	B	C
1			
2	内蒙古	呼和浩特	华北
3	广西	南宁	华南
4	西藏	拉萨	西南
5	新疆	乌鲁木齐	西北
6	宁夏	银川	西北
7			

图 8-13 查询结果直接发送到单元格

8.2.3 Field 对象

一个 RecordSet 对象有一个 Fields 字段集合成员，该集合表示结果记录集的所有列，对于上面自治区查询的实例，Fields 主要取决于 SQL 语句中 Select 子句后面列出的字段名称。

Field 表示其中一个字段，Field 对象有如下三大属性。

- ❑ Name：字段的名称。
- ❑ Value：该字段的当前取值，与 RecordSet 的游标有关。
- ❑ Type：字段的类型，也就是数据库设计时的字段类型。

下面的代码段基于上述“查询 Access 数据库”过程修改。

```
If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
    Debug.Print "结果记录集的记录总数：", rst.RecordCount
    Debug.Print "结果记录集的字段总数：", rst.Fields.Count
    For Each fld In rst.Fields
        Debug.Print fld.Name, fld.Type, fld.Value
    Next fld
    ActiveSheet.Range("A2").CopyFromRecordset rst
    Dim i As Integer
    For i = 0 To rst.Fields.Count - 1
        ActiveSheet.Cells(1, i + 1).Value = rst.Fields(i).Name
    Next i
    rst.Close '关闭 rst
End If
```

代码分析：当执行 SQL 查询后，根据 RecordSet 打印记录总数和字段总数，然后遍历每个字段的名称、类型和当前值。

最后一个 For 循环用来把字段名称加到 Excel 第一行单元格中，如果使用数字下标遍历 Fields，是从 0 开始的。

运行上述代码，Excel 中单元格的数据带有标题行，如图 8-14 所示。

立即窗口的打印结果如图 8-15 所示。

立即窗口中打印出的“省份名称”是一个字段的名称，202 表示字段类型是文本，它是一个 ADODB 内置枚举常量的等价整数，由于本实例中 3 个字段全是文本，所以打印出来的都是 202。在对象浏览器中输入 DataTypeEnum 执行搜索，可以看到所有字段类型的枚举常量，如图 8-16 所示。

F10

	A	B	C
1	省份名称	省会	区域
2	内蒙古	呼和浩特	华北
3	广西	南宁	华南
4	西藏	拉萨	西南
5	新疆	乌鲁木齐	西北
6	宁夏	银川	西北
7			

图 8-14　遍历字段并发送到单元格

立即窗口

```
结果记录集的记录总数:          5
结果记录集的字段总数:          3
省份名称            202        内蒙古
省会                202        呼和浩特
区域                202        华北
```

图 8-15　打印结果记录集的有关信息

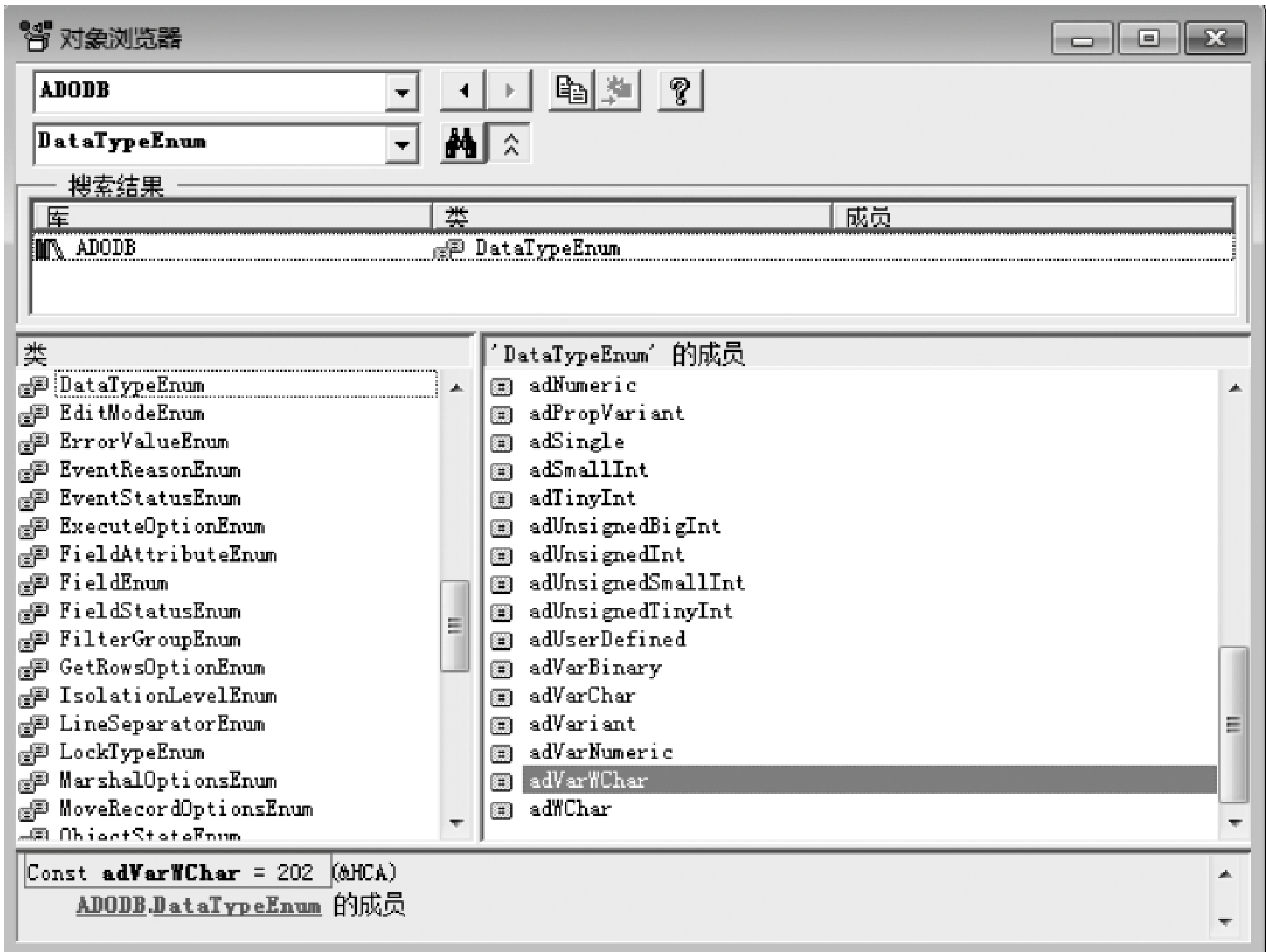


图 8-16　字段类型常量及其对应的数值

在立即窗口中的第 3 列中，可以看到打印出的记录是内蒙古的相关信息，这是为什么呢？能不能打印出其余 4 个少数民族自治区的信息呢？这就涉及如何遍历结果记录集中记录行的问题了。

8.2.4　遍历记录行

SQL 中使用 Select 子句进行查询，得到的结果记录集是符合条件的一个子表，那么执行 SQL 查询后，RecordSet 对象的默认游标处于结果记录集的第一条记录上。

如果要提取后面记录的数据，必须使用 Move 系列的方法移动游标，移动游标后，当前记录行（Current Record）的位置也随之改变。

- MoveFirst：移动到首条记录。
- MoveLast：移动到最末行。
- MovePrevious：移动到当前记录的前一条。
- MoveNext：移动到当前记录的后一条。
- Move：移动到特定的位置。

通过 Move 系列方法移动游标后，由于当前记录行位置发生变化，RecordSet.Fields(i). Value 也会变化，从而实现提取不同记录行的数据。

同时，使用 Move 系列方法时必须考虑是否移动到结构记录集的边界，如果当前记录是记录集的首条，那么再执行 MovePrevious 方法后，游标被移动到记录集以外，此时 RecordSet 的 BOF 属性为 True；同理，如果当前记录是记录集的最后一条，那么再执行 MoveNext 方法后，游标也被移动到记录集以外，此时 RecordSet 的 EOF 属性为 True。

因此，经常利用 EOF 属性配合 Do...Loop 循环来遍历所有记录行。

```
Sub 移动游标 ()
    Dim mark As ADODB.BookmarkEnum
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
        .Open
        If .State = ADODB.ObjectStateEnum.adStateOpen Then
            Set rst = New ADODB.Recordset
            rst.CursorLocation = adUseClient
            rst.Open Source:="Select 省份名称 , 省会 , 区域 From Detail Where 少数民
族自治区 =True", ActiveConnection:=cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset,
LockType:=ADODB.LockTypeEnum.adLockOptimistic
            If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                Debug.Print " 结果记录集的记录总数: ", rst.RecordCount
                Debug.Print " 结果记录集的字段总数: ", rst.Fields.Count
                rst.MoveFirst ' 第一条
                Debug.Print rst.Fields(" 省会 ").Value, rst.Fields(" 省份名称 ").Value
                rst.MoveNext ' 第二条
                Debug.Print rst.Fields(" 省会 ").Value, rst.Fields(" 省份名称 ").Value
                rst.MoveLast ' 最后一条 (第 5 条)
                mark = rst.Bookmark
                Debug.Print rst.Fields(" 省会 ").Value, rst.Fields(" 省份名称 ").Value
                rst.MovePrevious ' 前一条 (第 4 条)
                Debug.Print rst.Fields(" 省会 ").Value, rst.Fields(" 省份名称 ").Value
                rst.Move NumRecords:=-2, Start:=ADODB.BookmarkEnum.adBookmarkLast
                Debug.Print rst.Fields(" 省会 ").Value, rst.Fields(" 省份名称 ").Value
                rst.Close ' 关闭 rst
            End If
            .Close ' 关闭 cnn
        Else
            MsgBox " 失败！ ", vbExclamation
        End If
    End With
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

代码分析：无论是否使用 Move 系列方法，RecordSet 都是一个不变的子表，使用 Move 系列方法主要是改变 RecordSet 的当前活动行。

Move 系列方法中最难理解的是 Move。该方法可以设置两个参数，第一个参数 NumRecords 接受一个长整型的数字，表示从参照行引起的偏移，可以是正数或负数。第二

个参数 Start 接受一个枚举常量，用来设置参照行。

```
rst.Move NumRecords:=-2, Start:=ADODB.BookmarkEnum.adBookmarkLast
```

这行代码表示，以最末一行为参照，−2 表示向上移动 2 行，那就是移动到“西藏”那一行。运行上述过程，立即窗口的结果如图 8-17 所示。

```
立即窗口
结果记录集的记录总数:        5
结果记录集的字段总数:        3
呼和浩特        内蒙古
南宁            广西
银川            宁夏
乌鲁木齐        新疆
拉萨            西藏
```

图 8-17　游标的移动

另外，Start 参数除了使用内置枚举常量外，还可以用 RecordSet 的 Bookmark 属性作为参照起始行。

```
rst.Move NumRecords:=-2, Start:=mark
```

其中 mark 变量是之前计算出的一个书签值。

如果要按正常顺序打印结果记录集，一直使用 MoveNext 方法即可。下面的过程分别使用 For 循环和 Do...Loop 循环正序打印每条记录。

```
Sub 遍历记录行()
    Dim i As Integer
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
        .Open
        If .State = ADODB.ObjectStateEnum.adStateOpen Then
            Set rst = New ADODB.Recordset
            rst.CursorLocation = adUseClient
            rst.Open Source:="Select 省份名称,省会,区域 From Detail Where 少数民族自治区=True", ActiveConnection:=cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
            If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                rst.MoveFirst
                For i = 1 To rst.RecordCount
                    Debug.Print rst.Fields("省份名称").Value
                    rst.MoveNext
                Next i
                rst.MoveFirst
                Do Until rst.EOF
                    Debug.Print rst.Fields(1).Value
                    rst.MoveNext
                Loop
                rst.Close '关闭 rst
            End If
            .Close '关闭 cnn
        Else
            MsgBox "失败！", vbExclamation
        End If
    End With
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

需要注意的是，不管是哪一种循环方式，循环之前请把游标移动到第一行。在使用 Do...Loop 循环结构时，循环体内部必须有 RecordSet.MoveNext 语句，否则当前记录行固定在同一个位置，使得 EOF 属性一直是 False，陷入死循环。

8.2.5 使用 Connection.Execute 方法执行 SQL 语句

SQL 查询中，Select 子句从数据库中查询出符合条件的记录，形成结果记录集，并不破坏原数据表的数据。

还有一些子句，例如 Insert Into、Delete、Update 子句的作用和目的并非查询，而是对数据库中原数据表的增加、删除和修改更新操作，此时不需要返回结果记录集。也就是说，操作数据库的目的不是查询，而是修改原表，无须使用 RecordSet 对象，仅仅使用 Connection 对象的 Execute 方法即可执行 SQL 语句。

下面的过程将删除数据表 Detail 中省会为空的记录整行。

```
Sub Execute方法查询Access数据库()
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
        .Open
        If .State = ADODB.ObjectStateEnum.adStateOpen Then
            .Execute CommandText:="Delete From Detail Where 省会 Is Null"
            MsgBox "成功执行SQL语句。", vbInformation
            .Close '关闭cnn。
        Else
            MsgBox "失败！", vbExclamation
        End If
    End With
    Set cnn = Nothing
End Sub
```

执行上述过程后，在 Access 中再次打开数据库 ChinaProvince，可以看到表 Detail 中省会为空白的记录被删除。

因此，在实际编程过程中，要根据执行 SQL 的目的来决定是否使用 RecordSet 对象。

8.2.6 使用 Command.Execute 方法执行 SQL 语句

Command 对象也有一个 Execute 方法用于执行 SQL 语句，使用 Command 对象之前，先创建一个 Connection 对象，当 Connection 对象正常 Open 之后，让 Command 对象的 ActiveConnection 属性等于这个 Connection 对象即可。

下面的过程使用 Command 对象执行删除查询。

```
Sub Command方法查询Access数据库()
    Dim cmd As ADODB.Command
    Set cnn = New ADODB.Connection
    Set cmd = New ADODB.Command
```

```
        With cnn
             .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
             .Open
             Set cmd.ActiveConnection = cnn
             If .State = ADODB.ObjectStateEnum.adStateOpen Then
                 cmd.CommandType = adCmdText
                 cmd.CommandText = "Delete From Detail Where 区域='华北'"
                 cmd.Execute '删除区域为华北的所有记录
                 .Close '关闭 cnn
             End If
        End With
        Set cnn = Nothing
    End Sub
```

8.3　窗体中显示查询结果

通过 SQL 中的 Select 子句得到的结果记录集，可以发送到 Excel 单元格区域，也可以遍历记录行，把每行记录的信息打印到立即窗口。在实际开发过程中，数据查询的结果经常显示于窗体控件中。

在窗体上呈现查询结果时，既可以一次显示结果记录集的所有行，也可以一次只显示一行，根据任务需要而定，如果一次显示所有记录，使用 ActiveX 控件中的 DataGrid 或者 MSForms 中的 ListBox 控件比较适合，一次显示一条记录的情形，使用 TextBox 即可。

8.3.1　ListBox 控件显示结果记录集

下面的实例在 VBA 窗体上放置一个列表框来显示人口大于 5000 万人的省份。

VBA 窗体上放置一个 TextBox，用来输入 SQL 查询语句，再放置一个 ListBox，用来显示结果记录集。其中文本框的 KeyDown 事件代码如下。

```
    Private Sub TextBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal
Shift As Integer)
        If KeyCode = vbKeyReturn Then
        Set cnn = New ADODB.Connection
        With cnn
             .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
             .Open
             If .State = ADODB.ObjectStateEnum.adStateOpen Then
                 Set rst = New ADODB.Recordset
                 rst.CursorLocation = adUseClient
                  rst.Open Source:=Me.TextBox1.Text, ActiveConnection:=cnn, CursorType:=
ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
                 If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                     Me.ListBox1.ColumnCount = rst.Fields.Count
                     ActiveSheet.UsedRange.ClearContents
                     Range("A2").CopyFromRecordset rst
                     Dim i As Integer
```

```
                For i = 0 To rst.Fields.Count - 1
                    ActiveSheet.Cells(1, i + 1).Value = rst.Fields(i).Name
                Next i
                Me.ListBox1.Column = Application.WorksheetFunction.Transpose
(Range("A1").Resize(rst.RecordCount+1, rst.Fields.Count).Value)
                rst.Close '关闭 rst
            End If
            .Close '关闭 cnn
        End If
    End With
    Set rst = Nothing
    Set cnn = Nothing
    End If
End Sub
```

代码分析：当在文本框中按下回车键时，执行 SQL 查询，查询结束后，首先把结果记录集发送至单元格区域，然后设置列表框的 Column 属性为 Range 即可。需要注意的是，Excel 中的数据是带有标题行的，也就是数据条目比结果记录集多 1，因此向 ListBox 赋值的时候，通过 Resize 方法更改区域尺寸为 Range("A1").Resize(rst.RecordCount+1, rst.Fields.Count).Value。

启动窗体，在文本框中输入 SQL 查询语句，按下回车键，可以看到列表框中显示与单元格中显示内容是完全一样的，如图 8-18 所示。

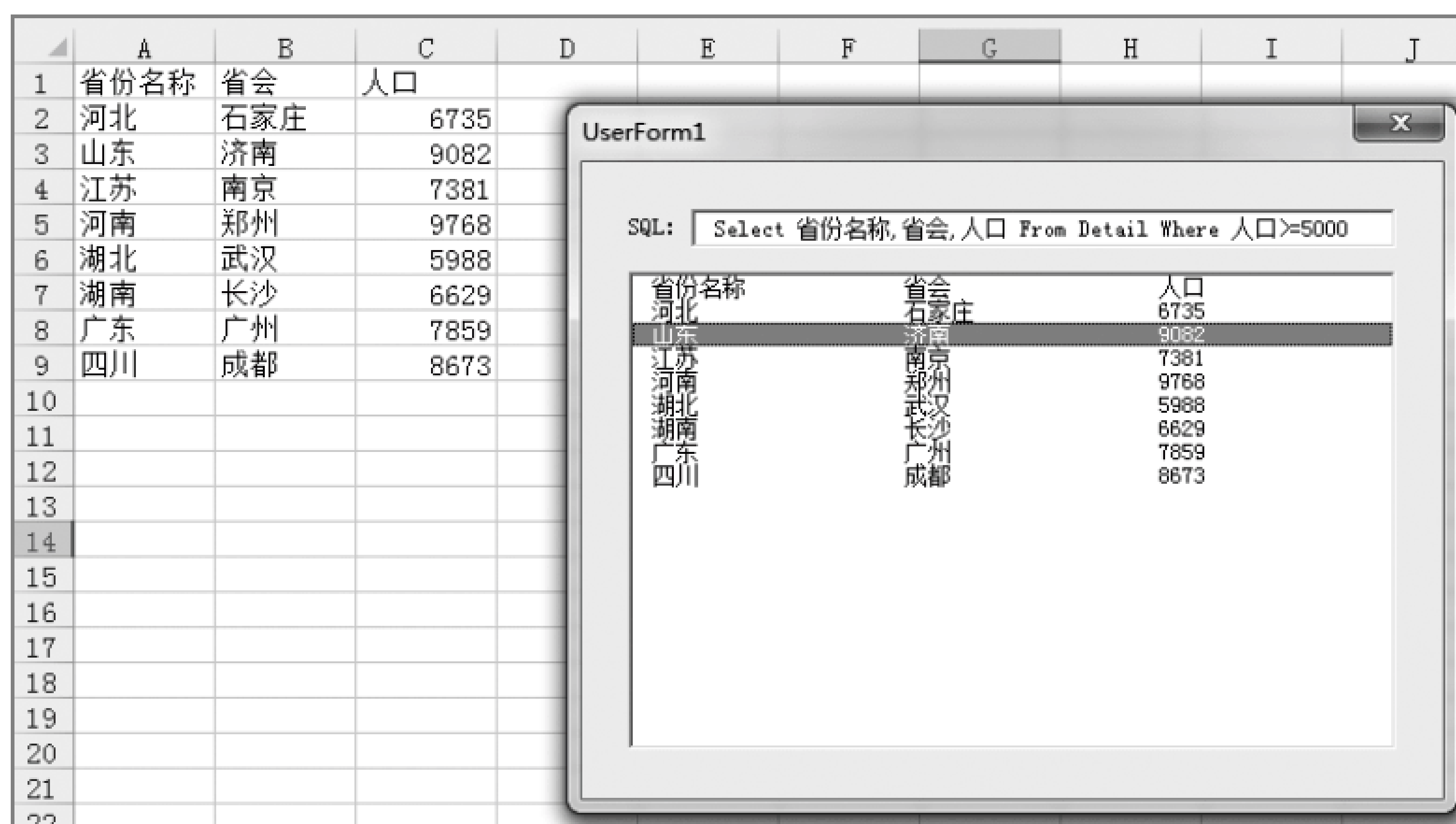


图 8-18 VBA 的列表框控件中显示查询结果

8.3.2 使用 TextBox 控件显示单条记录

下面的实例在用户窗体上放置 6 个 TextBox 和一个 CheckBox，复选框控件用于显示是否为少数民族自治区。

在窗体右侧添加 4 个 CommandButton 控件，用来移动游标以切换记录行。

最上面的文本框用来输入 SQL 语句并按下回车键进行执行，具体 KeyDown 事件如下。

```
Private Sub TextBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal
Shift As Integer)
```

```
        If KeyCode = vbKeyReturn Then
            Set cnn = New ADODB.Connection
            With cnn
                    .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=" & ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
                .Open
                If .State = ADODB.ObjectStateEnum.adStateOpen Then
                    Set rst = New ADODB.Recordset
                    rst.CursorLocation = adUseClient
                    rst.Open Source:=Me.TextBox1.Text, ActiveConnection:=cnn, CursorType:=
ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
                End If
            End With
        End If
    End Sub
```

需要注意的是，由于本实例需要经常移动游标，因此窗体显示期间不能关闭 RecordSet 和 Connection 对象，并且这两个对象要声明为模块级变量。

接下来为 4 个导航按钮设置单击事件，如下所示。

```
    Private Sub CommandButton1_Click()
        '第一条
        rst.MoveFirst
        Call GetData
    End Sub

    Private Sub CommandButton2_Click()
        '后一条
        If rst.EOF = False Then
        rst.MoveNext
        Call GetData
        End If
    End Sub

    Private Sub CommandButton3_Click()
        '前一条
        If rst.BOF = False Then
        rst.MovePrevious
        Call GetData
        End If
    End Sub

    Private Sub CommandButton4_Click()
        '最后一条
        rst.MoveLast
        Call GetData
    End Sub
```

需要注意的是，RecordSet 执行 MoveNext 方法时，一定要检查 EOF 属性是否为 True；MovePrevious 方法执行之前要检查 BOF 属性是否为 True。当游标成功移动后，调用 GetData 过程，把记录集的数据放入对应文本框中。

```
Sub GetData()
    If rst.BOF Or rst.EOF Then

    Else
        With Me
            .TextBox2.Text = rst.Fields("省份名称").Value & ""
            .TextBox3.Text = rst.Fields("简称").Value & ""
            .TextBox4.Text = rst.Fields("省会").Value & ""
            .TextBox5.Text = rst.Fields("区域").Value & ""
            .TextBox6.Text = rst.Fields("面积").Value & ""
            .TextBox7.Text = rst.Fields("人口").Value & ""
            .CheckBox1.Value = rst.Fields("少数民族自治区").Value & ""
        End With
    End If
End Sub
```

代码分析：在使用 RecordSet 的 Fields 提取数据前，需要保证 BOF 和 EOF 属性均为 False 才行。当某个字段的值是 Null（数据为空白），此时访问其 Value 属性会出错，故此后面均加上 & ""。

启动用户窗体，在最上面的文本框中输入 SQL 语句并按下回车键，然后单击“最后一条”按钮，左侧各个文本框显示相应信息，如图 8-19 所示。

图 8-19 文本框中显示结果

以上内容的源代码文件为“实例文档 45.xlsm”。

8.3.3 使用 DataGrid 控件显示结果记录集

显示结果记录集最适合的控件要数 Microsoft DataGrid Control 6.0，该控件以表格的形式显示结果记录集，并且可以自动移动游标、自动修改、删除记录和增加记录。

但是很多情况下不能向 VBA 的 UserForm 中插入 DataGrid 控件，因此下面的实例向 VB6 的 Form 中添加一个 Text 文本框和一个 DataGrid 控件。

DataGrid 控件不是 VB6 的基本控件，因此需要单击 VB6 的菜单【工程 / 部件】，勾选“Microsoft DataGrid Control 6.0”，如图 8-20 所示。



图 8-20　窗体上添加 DataGrid 控件

在窗体的设计模式下，可以把 DataGrid 控件从控件工具箱中拖放到窗体上。

窗体的代码视图中，文本框的 KeyDown 事件用来连接数据库、执行查询并把查询的结果记录集绑定到 DataGrid 控件。

```
    Private cnn As ADODB.Connection, rst As ADODB.Recordset
    Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
        If KeyCode = vbKeyReturn Then
            Set cnn = New ADODB.Connection
            With cnn
                .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=
" & App.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
                .Open
                If .State = ADODB.ObjectStateEnum.adStateOpen Then
                    Set rst = New ADODB.Recordset
                    rst.CursorLocation = adUseClient
                    rst.Open Source:=Me.Text1.Text, ActiveConnection:=cnn, CursorType:=
ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
                    With Me.DataGrid1
                        .AllowAddNew = True          '允许添加新记录
                        .AllowDelete = True          '允许删除记录
                        .AllowUpdate = True          '允许修改记录
                        Set .DataSource = rst
                    End With
                End If
            End With
        End If
    End Sub
```

代码分析：可以看出，与 VBA 中使用 ADO 唯一不同的是 With Me.DataGrid 那 4 行代码，前 3 行用来设置 DataGrid 是否允许删除、添加、修改记录。

Set Me.DataGrid.DataSource = rst 这句是关键，作用是把结果记录集绑定到控件。这也是这个表格控件能够显示数据的原因。

启动窗体，在文本框中输入 SQL 语句，并按下回车键，DataGrid 控件中显示相应数据。这样就制作出了一款 SQL 语句测试器，如图 8-21 所示。



图 8-21　DataGrid 控件中显示结果记录集

使用 DataGrid 控件，不需要遍历字段、记录行，就可以完美显示查询出的结果，而且可以像在 Access 中一样通过 DataGrid 控件编辑、修改数据表。

以上实例的 VB 工程源代码为“UseDataGrid.vbp”。

基于 ADO 技术进行数据库的读写，可以制作很多种应用程序，笔者以 Access 数据库作为容器，使用 VB6 结合 DataGrid 控件制作了一款象棋棋谱浏览器，可以方便地存谱、读谱，如图 8-22 所示。

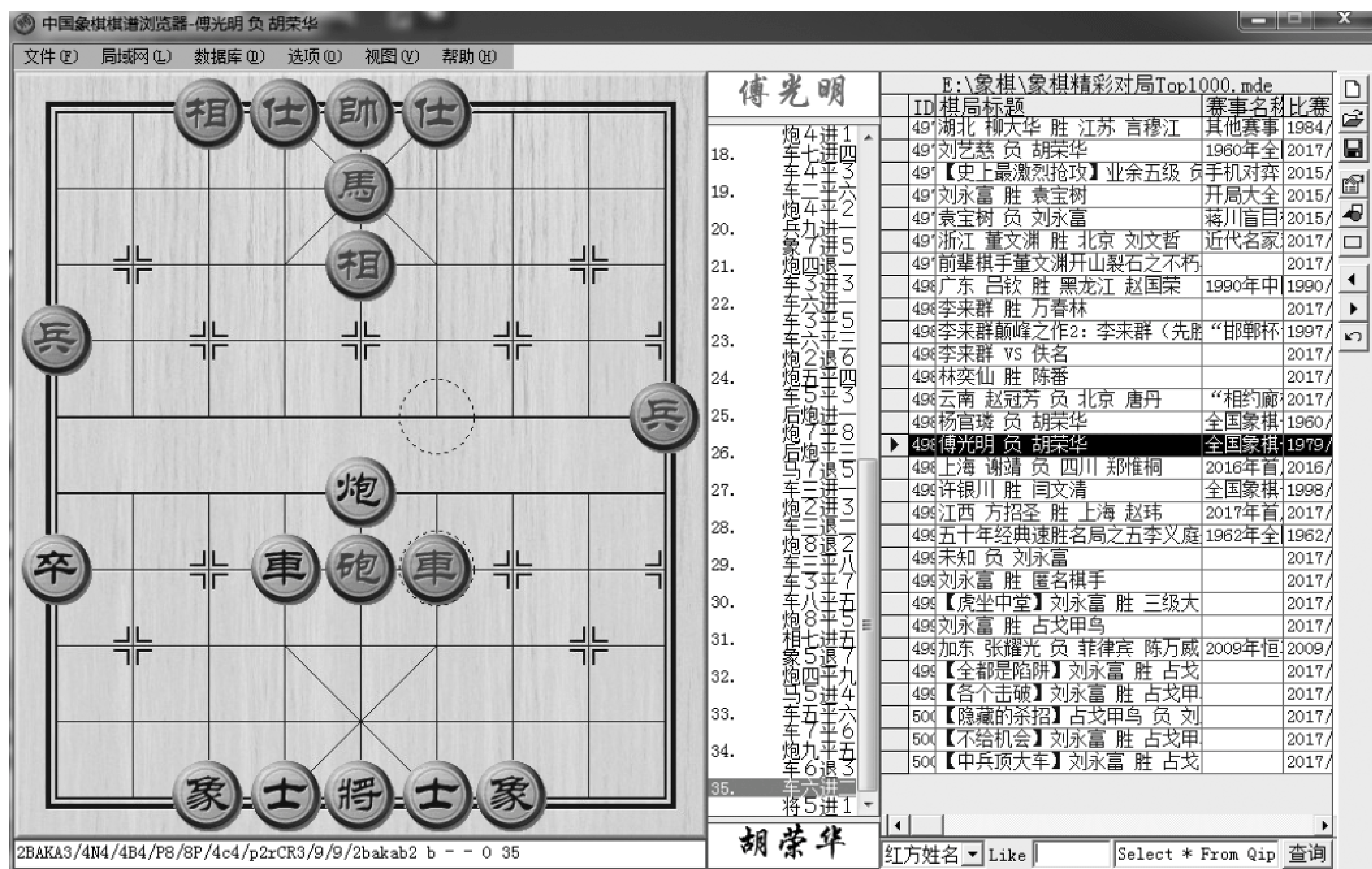


图 8-22　VB 联用 Access 数据库制作的软件

对本产品感兴趣的读者可以从本书配套资源下载 CnChessQipu-Setup-20170427.exe。

8.4　SQL 结构化查询语言详解

SQL 是目前各种关系数据库系统中广泛采用的标准语言，ADO 对象通过 SQL 来操作数据库中的数据。

前面已经讲述过，ADO 使用执行 SQL 查询的方法有 3 个：Connection 对象直接 Execute 的方法适合于不需要返回结果记录集的场合；RecordSet 的 Open 方法适合于返回结果记录集；Command 的 Execute 的方法类似于 Connection 对象。

实际上，数据库编程的难点和重点在于 SQL 语句的构造、理解和运用上，前面已经涉及使用 Select 语句的简单用法，但由于 SQL 语句是一个字符串，书写过程中无任何语法提示，因此本节讲解查询语句最常用的构造方法。

操作数据表中数据的 SQL 语句主要如下。

- Select 子句：从表中找到符合条件的记录，或者对数据表进行信息统计。该子句产生结果记录集，不破坏原数据表。
- Insert 子句：向原数据表中插入新数据记录。
- Update 子句：修改原数据表中特定的记录。
- Delete 子句：删除原数据表中特定的记录。
- Where 子句：配合以上 4 个语句执行筛选。

此外，还有一些用于创建、修改数据库、数据表的 SQL 语句，将在 8.5 节讲解。

8.4.1　使用 Select 语句查询

Select 语句的作用是从原数据表查找符合特定条件的记录，并作为结果记录集返回。使用 Select 语句可以实现行的筛选、列的筛选、记录汇总、结果记录排序、分类汇总等功能。

Select 语句的一般语法如下。

```
Select 字段 From 表 Where 条件
```

为了便于讲解，此处仍然采用前面的 ChinaProvince 数据库，以 DataGrid 控件显示结果记录集。

1. 字段筛选

所有 SQL 语句中，字段名称直接书写即可，不能用引号括起来。Select 后面的字段列表用半角逗号隔开，如果是原数据表中的所有字段，使用 * 即可。

查询出的结果记录集，一般仍然保持原数据表中的字段名称，当然也可以使用 As 子句改名。例如下面这句。

```
Select 省份名称，省会+'市' As Capital, 人口 As Population From Detail
```

这句 SQL 语句的功能是从原数据表中列出【省份名称】【省会】后面加“市”，并且新字段名为 Capital、【人口】的新名称为 Population。查询结果如图 8-23 所示。



省份名称	Capital	Population
黑龙江	哈尔滨市	3813
吉林	长春市	2699
辽宁	沈阳市	4203
北京	北京市	1423
天津	天津市	1007
河北	石家庄市	6735
山西	太原市	3294
内蒙古	呼和浩特市	2379
山东	济南市	9082
江苏	南京市	7381
安徽	合肥市	
上海	上海市	1625
浙江	杭州市	4647
江西		4222
福建	福州市	3466
河南	郑州市	9768
湖北	武汉市	5988

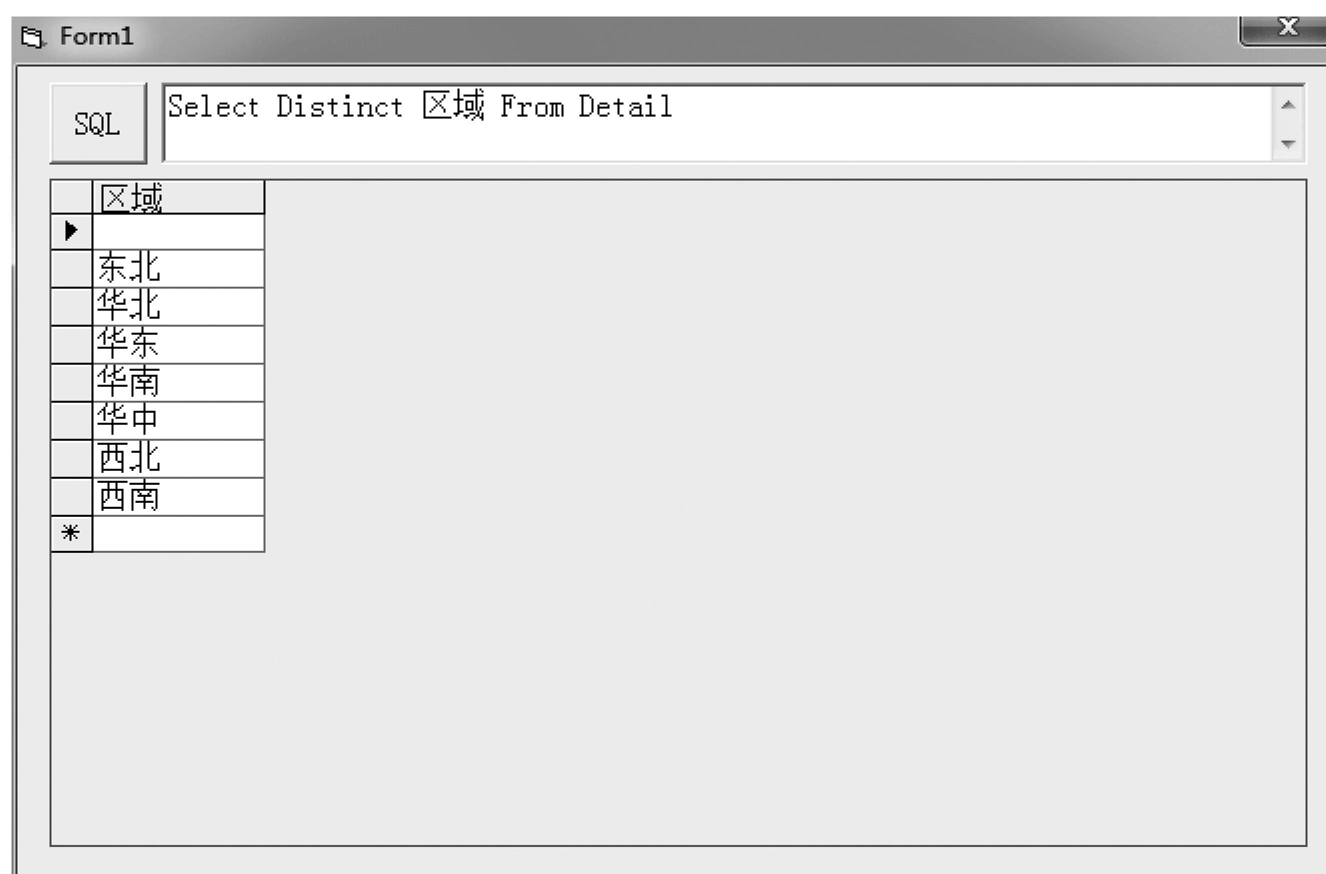
图 8-23　显示部分字段

再次强调，使用 Select 子句查询后，返回的是新的结果记录集，不会损害到原数据表。

2. 使用 Distinct 提取唯一字段

数据表的每列数据中经常有内容一样的数据，例如要查看 ChinaProvince 数据库中有哪些不同的区域呢？

下面这句代码是对【区域】字段中的内容进行去重，结果如图 8-24 所示。



区域
东北
华北
华东
华南
华中
西北
西南

图 8-24　显示某字段的唯一值

```
Select Distinct 区域 From Detail
```

可以看到结果记录集共 1 列，8 行（包含 1 个空白行，这是因为原数据表中有的记录没有填写【区域】）。

这个去除重复的功能看上去类似于第 7 章讲过的字典。

字段去重的 VBA 代码如下。

```
Public cnn As ADODB.Connection, rst As ADODB.Recordset, fld As ADODB.Field
Sub 字段去重()
    Set cnn = New ADODB.Connection
    With cnn
```

```
            .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
            .Open
            If .State = ADODB.ObjectStateEnum.adStateOpen Then
                Set rst = New ADODB.Recordset
                rst.CursorLocation = adUseClient
                rst.Open Source:="Select Distinct 区域 From Detail", ActiveConnection:=
cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.
adLockOptimistic
                If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                    Debug.Print "结果记录集的记录总数: ", rst.RecordCount
                    Debug.Print "结果记录集的字段总数: ", rst.Fields.Count
                    Dim i As Integer
                    For i = 1 To rst.RecordCount
                        Debug.Print rst.Fields("区域").Value
                        rst.MoveNext
                    Next i
                    rst.Close                   '关闭 rst
                End If
                .Close                          '关闭 cnn
            Else
                MsgBox "失败! ", vbExclamation
            End If
        End With
        Set rst = Nothing
        Set cnn = Nothing
    End Sub
```

代码分析：连接数据库的过程和以前没有变化，要注意看的是 rst.Open 那句，以及使用 For 循环打印记录的那部分。

运行上述过程，立即窗口打印出了唯一的【区域】，如图 8-25 所示。

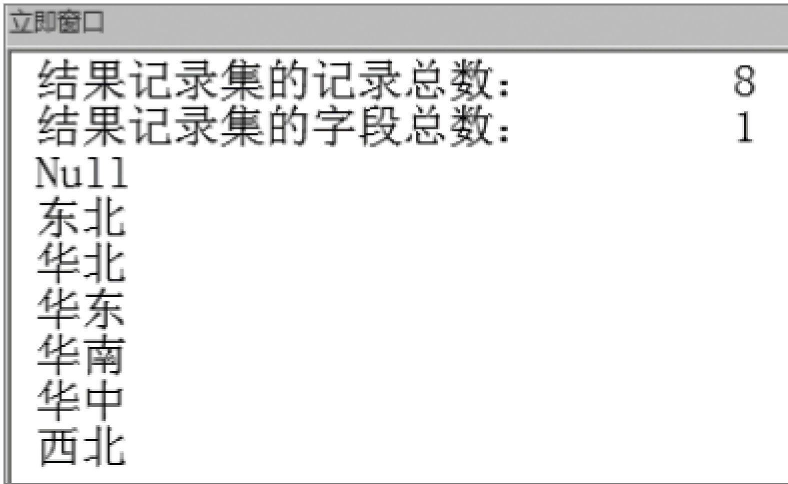


图 8-25　打印区域字段的唯一值

3. 使用 Top 提取前几条记录

下面这句代码从数据表中选择前 10 条，结果如图 8-26 所示。



省份名称	简称	省会	区域	面积	人口	少数民族
黑龙江	黑	哈尔滨	东北	45.48	3813	0
吉林	吉	长春	东北	18.74	2699	0
辽宁	辽	沈阳	东北	15.59	4203	0
北京	京	北京	华北	1.68	1423	0
天津	津	天津	华北	1.13	1007	0
河北	冀	石家庄	华北		6735	0
山西	晋	太原	华北	15.63	3294	0
内蒙古	蒙	呼和浩特	华北	118.3	2379	-1
山东	鲁	济南	华东	15.38	9082	0
江苏	苏	南京	华东	10.26	7381	0

图 8-26　查询前 n 条记录

```
Select Top 10 * From Detail
```

下面这句代码从数据表中选择前 10% 的记录，如果原数据表有 34 条记录，那么前 10% 就是前 4 条记录，结果图略，读者可自行验证。

```
Select Top 10 Percent * From Detail
```

8.4.2 使用 Where 子句进行记录筛选

利用 Where 子句可以设定条件，用来选择性地筛选记录行。

Where 子句的难点是条件表达式的构造，一般的条件构造格式如下。

```
Where 字段 比较运算符 常量表达式
```

例如，Where 区域 =' 东北 '，就是只把【区域】为东北的记录筛选出来，结果如图 8-27 所示。



图 8-27　使用 Where 作为筛选条件

SQL 中的不等于使用 <> 表示。例如，Where ' 西北 '<> 区域，表示【区域】不是西北的记录。

由于比较的基准“西北”是一个常量字符串，因此一定要用单引号括起来。比较基准如果是日期，要用两个 # 括起来，比较基准是数字或者是布尔常量，则不需要括起来。

一些常用 Where 子句如下。

- 大于指定日期：Where 出生日期 >=#1992/4/6#。
- 两门成绩总和：Where 语文 + 数学 >=120。
- 介于范围之间：Where 人口 Between 2000 And 4000。
- 多个数值之一：Where 区域 In (' 东北 ',' 西北 ',' 西南 ')。
- 数据为空：Where 省会 Is Null。

1. 使用 And 和 Or 多条件组合

Where 子句还允许使用 And、Or 进行多个条件的组合。

使用 And 把多个条件连接起来，筛选出每个条件都成立的记录。例如：

```
Where 面积 >10 And 人口 <2000
```

把面积大于 10 并且人口小于 2000 万人的记录筛选出来。

```
Where 语文 <60 Or 数学 <60
```

只要有一科不及格的就列出来。

2. 使用 Not 实现条件的否定

SQL 语句中的 Not 关键字功能很强大，通常置于条件之前，起到否定该条件的作用。

例如：Where Not 人口 <6000，相当于 Where 人口 >=6000。

Where Not 简称 Is Null，表示【简称】字段非空的记录。

Not 不仅可以置于整个条件之前，还可以置于比较关键字之前，例如：Where 简称 Is Not Null 等价于 Where Not 简称 Is Null。Where 区域 Not In (' 东北 ',' 西北 ') 等价于 Where Not 区域 In (' 东北 ',' 西北 ')。Where 省会 Not Like '% 州 ' 等价于 Where Not 省会 Like '% 州 '。

3. 使用 Like 模糊匹配

与 VBA 中的 Like 类似，SQL 语句中的 Like 也可以模糊匹配字段中的内容。

Like 语法格式如下。

```
Where 字段 Like 含通配符的表达式
```

可以使用的通配符有如下几种情况。

- %：百分号可以匹配任意多个任意字符。
- _：一个下画线可以匹配一个任意字符。
- [你我他]：可以匹配你、我、他三个字中的任一个。
- [A-G]：可以匹配 A ~ G 之内的任一个。
- [!A-G]：可以匹配 A ~ G 之外的任一个字符，感叹号表示否定。

例如：

```
Select * From Detail Where 省份名称 Like '%西' And 区域 Like '华%'
```

表示匹配以“西”结尾的【省份名称】，并且以“华”开头的【区域】，结果如图 8-28 所示。

图 8-28　使用 Like 和通配符进行筛选

再例如：Where 省份名称 Like '[! 山陕] 西 '，可以匹配到江西、广西，但是不能匹配到陕西、山西。

8.4.3 使用 Order By 进行排序

Select 语句中可以包含 Order By 子句以对结果记录集进行排序。具体语法格式如下。

```
Order By 字段或表达式 ASC 或 DESC
```

其中，ASC 是升序，DESC 是降序。Order By 子句可以包含一个以上的排序基准，如果第 1 个基准比较不出大小，则按第 2 基准，以此类推。

针对数据记录的比较，如果是中文比较，则按照拼音顺序，数字和日期按照大小；如果记录中包含空值（Null）则认为是最小。

例如：

```
Select * From Detail Order By 区域 ASC, 人口 DESC
```

表示按照区域升序，当区域相同时按照人口降序排列，结果如图 8-29 所示。

Form1

SQL　Select * From Detail Order By 区域 ASC, 人口 DESC

省份名称	简称	省会	区域	面积	人口	少数民族
辽宁	辽	沈阳	东北	15.59	4203	0
黑龙江	黑	哈尔滨	东北	45.48	3813	0
吉林	吉	长春	东北	18.74	2699	0
河北	冀	石家庄	华北		6735	0
山西	晋	太原	华北	15.63	3294	0
内蒙古	蒙	呼和浩特	华北	118.3	2379	-1
北京	京	北京	华北	1.68	1423	0
天津	津	天津	华北	1.13	1007	0
山东	鲁	济南	华东	15.38	9082	0
江苏	苏	南京	华东	10.26	7381	0
浙江	浙	杭州	华东	10.2	4647	0
江西	赣		华东	16.7	4222	0
福建	闽	福州	华东	12.13	3466	0
安徽	皖	合肥	华东	13.97		0
广东	粤	广州	华南	18	7859	0
广西	桂	南宁	华南	23.6	4822	-1
海南省	琼	海口	华南	3.4	803	0

图 8-29　结果记录集的排序

此外，排序的关键字还可以是含字段名称的表达式。例如，按各省份人口密度排序，就需要用【面积】除以【人口】得到人均面积。

下面的 SQL 语句把【面积】/【人口】的结果字段重命名为【人均面积】，然后按人均面积降序排列。

```
Select 省份名称，省会，面积，人口，面积/人口 As 人均面积 From Detail Order By 面积/人口 DESC
```

可以看到西藏和青海的人均面积最大，地广人稀，结果如图 8-30 所示。

注意　如果 Select 子句中同时使用 Where 和 Order By，则需要把 Where 子句放在 Order By 之前。

Form1

SQL　Select 省份名称,省会,面积,人口,面积/人口 As 人均面积 From Detail Orcer By 面积/人口　DESC

省份名称	省会	面积	人口	人均面积
西藏	拉萨	122.8	267	.459925105062763
青海	西宁	72.23	529	.136540649067927
新疆	乌鲁木齐	166	1906	.087093389296957
内蒙古	呼和浩特	118.3	2379	4.97267772390743E-02
台湾	台湾	3.6	227	1.58590304168836E-02
宁夏	银	6.64	527	1.25996202400106E-02
黑龙江	哈尔滨	45.48	3813	1.19276159303006E-02
甘肃	兰州	45.44	3813	1.19171252627089E-02
云南	昆明	38.33	4333	8.84606550451297E-03
吉林	长春	18.74	2699	6.94331225310047E-03
陕西	西安	20.56	3647	5.6375101359864E-03
四川	成都	48.14	8673	5.55055913635979E-03
广西	南宁	23.6	4822	4.89423483647236E-03
山西	太原	15.63	3294	4.74499092727411E-03
贵州	贵阳	17.6	3837	4.58691696155062E-03
海南省	海口	3.4	803	4.23412216110515E-03
江西		16.7	4222	3.95547152130257E-03

图 8-30　多列的计算结果作为排序基准

8.4.4　使用 Group By 进行分类汇总

与 Excel 中的 5 个汇总函数类似，SQL 中也允许使用如下 5 个汇总函数对记录之间进行汇总。

- ❑ Min：字段的最小值。
- ❑ Max：字段的最大值。
- ❑ Sum：字段的总和。
- ❑ Avg：字段的平均值，Null 值不统计。
- ❑ Count：非空记录奇数，Null 值不统计。

例如，下面的 SQL 语句对 34 个省份的面积进行汇总（数据表中有一个省份的面积为空）。

```
Select Sum(面积) As 总面积,Avg(面积) As 平均面积, Count(面积) As 计数 From Detail
```

查询结果如图 8-31 所示。

Form1

SQL　Select Sum(面积) As 总面积,Avg(面积) As 平均面积, Count(面积) As 计数 From Detail

总面积	平均面积	计数
946.9950	28.69681	33

图 8-31　使用汇总函数

从查询结果可以看出，Sum 等于 Avg * Count。

以上实例是对数据表的总体汇总，但很多情况下，需要按某字段进行分类汇总，例如，按照【区域】类别各自汇总，统计一下华北地区各省份人口总数等。类似于这种需求，需要使用 Group By 子句对字段进行分组，然后配合汇总函数。

下面的 SQL 语句按【区域】汇总【人口】。

```
Select 区域,Sum(人口) As 区域人口 From Detail Group By 区域
```

查询结果如图 8-32 所示。

Form1

SQL　Select 区域,Sum(人口) As 区域人口 From Detail Group By 区域

区域	区域人口
	1625
东北	10715
华北	14838
华东	28798
华南	14441
华中	22385
西北	10422
西南	20217

图 8-32　分类汇总

如果对汇总的组进行筛选，此时不使用 Where 子句，而是使用 Having 子句。

例如，只汇总区域人口大于 20000 万人的，那需要在结尾加上 Having Sum(人口)>20000。

```
Select 区域,Sum(人口) As 区域人口 From Detail Group By 区域 Having Sum(人口)>20000
```

查询结果如图 8-33 所示。

Form1

SQL　Select 区域,Sum(人口) As 区域人口 From Detail Group By 区域 Having Sum(人口)>20000

区域	区域人口
华东	28798
华中	22385
西南	20217

图 8-33　汇总结果作为筛选条件

8.4.5　使用 Select Into 语句把查询结果存入新表

前面已经讲过，使用 Select 语句一般会返回一个 RecordSet 结果记录集对象，如果想把返回的查询结果直接存入数据库，可以在 Select 子句的 From 子句之前加入 Into 表名。

下面的代码把 Select 语句的查询结果存入“人口分类汇总”表。

```
Sub 查询结果存表()
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
        .Open
        If .State = ADODB.ObjectStateEnum.adStateOpen Then
```

```
            .Execute "Select 区域,Sum
(人口) As 区域人口 Into 人口分类汇总 From Detail
Group By 区域"
        End If
    End With
    Set cnn = Nothing
End Sub
```

运行上述过程，手工打开 Access 数据库，可以看到多了一个“人口分类汇总”的表，如图 8-34 所示。

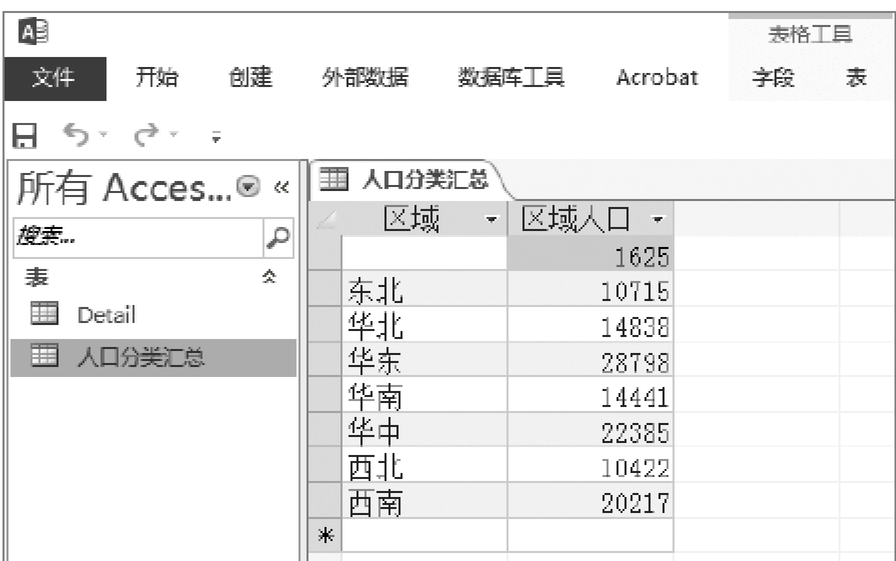


图 8-34　使用 SQL 语句把结果记录集保存为新表

8.4.6　使用 Insert Into 语句增加记录

Insert Into 语句可以向表中添加新记录。

例如，在 VBA 中，Execute 如下 SQL 代码，可以向 Detail 表中增加一条新记录。

```
.Execute "Insert Into Detail Values('象牙省','象','象牙市','东华区',200,320,
False)"
```

重新在 Access 中打开 Detail 表，可以看到最下面多了一行记录，如图 8-35 所示。



图 8-35　增加记录

以上这条 SQL 语句执行一次，增加一条，如果要添加大量新记录，需要把上述语句放入循环结构中。

可以看到，Insert Into...Values 的括号中是新记录的数据，如果数据是字符串，需要加上单引号，因此在实际编程过程中，构造这条语句相当麻烦，一般使用 RecordSet.AddNew 方法增加新记录。

与 Insert Into 相比，RecordSet 的 AddNew 方法不需要构造复杂的 SQL 语句，只需要规定各个字段的具体取值即可。下面的过程向 Detail 表增加两条新记录。

```
Sub 增加新记录()
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\ChinaProvince.accdb;Persist Security Info=False;"
        .Open
    End With
```

```
        Set rst = New ADODB.Recordset
        With rst
            .CursorLocation = adUseClient
            .Open Source:="Detail", ActiveConnection:=cnn, CursorType:=ADODB.Cursor
TypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
            .AddNew
            .Fields("省份名称").Value = "新省份1"
            .Fields("简称").Value = "新"
            .Fields("省会").Value = "新省会1"
            .Fields("区域").Value = "新区域1"
            .Fields("人口").Value = 230
            .Fields("少数民族自治区").Value = False
            .Update
            .AddNew
            .Fields("省份名称").Value = "新省份2"
            .Fields("简称").Value = "新"
            .Fields("省会").Value = "新省会2"
            .Fields("区域").Value = "新区域2"
            .Fields("人口").Value = 270
            .Fields("少数民族自治区").Value = True
            .Update
            .Close
        End With
        cnn.Close '关闭cnn
        Set rst = Nothing
        Set cnn = Nothing
    End Sub
```

代码分析：规定完各个字段的取值后，要用 RecordSet 的 Update 方法才能把数据存入数据库中。运行上述代码，可以看到 Detail 表下面多了两行，如图 8-36 所示。

Detail

省份名称	简称	省会	区域	面积	人口	少数民族自
重庆	渝	重庆	西南	8.23	3107	☐
西藏	藏	拉萨	西南	122.8	267	☑
新疆	新	乌鲁木齐	西北	166	1906	☑
甘肃	甘、陇	兰州	西北	45.44	3813	☐
青海	青	西宁	西北	72.23	529	☐
陕西	陕、秦	西安	西北	20.56	3647	☐
宁夏	宁	银川	西北	6.64	527	☑
新省份1	新	新省会1	新区域1	0	230	☐
新省份2	新	新省会2	新区域2	0	270	☑
*				0	0	☐

图 8-36　添加多条记录

8.4.7　使用 Delete 语句删除记录

Delete 语句用于从数据表中删除记录，一般和 Where 子句配合使用，可以选择性地删除。

例如，下面的 SQL 语句可以删除原数据表中所有【区域】为西南的记录。

```
Delete From Detail Where 区域='西南'
```

如果后面不带 Where 子句，则会清空数据表的所有记录。

8.4.8　使用 Update 语句修改记录

Update 语句用来修改更新原数据表中的特定记录数据。语法格式如下。

```
Update 表名 Set 字段名1=值1,字段名2=值2 Where 子句
```

下面的 SQL 语句把【区域】为西北的各省份的省会追加一个“市”字，然后把面积都设置为 100。

```
Update Detail Set 省会=省会+'市',面积=100 Where 区域='西北'
```

执行上述 SQL 语句后，在 Access 中可以看到 Detail 表中有 5 行记录被修改，如图 8-37 所示。

Detail

省份名称	简称	省会	区域	面积	人口	少数民族自
江苏	苏	南京	华东	10.26	7381	☐
安徽	皖	合肥	华东	13.97		☐
上海		上海		.63	1625	☐
浙江	浙	杭州	华东	10.2	4647	☐
江西	赣		华东	16.7	4222	☐
福建	闽	福州	华东	12.13	3466	☐
河南	豫	郑州	华中	16.7	9768	☐
湖北	鄂	武汉	华中	18.59	5988	☐
湖南	湘	长沙	华中	21.18	6629	☐
广东	粤	广州	华南	18	7859	☐
广西	桂	南宁	华南	23.6	4822	☑
海南省	琼	海口	华南	3.4	803	☐
香港	港	香港	华南	.11	686	☐
澳门	澳	澳门	华南	.025	44	☐
台湾	台	台湾	华南	3.6	227	☐
新疆	新	乌鲁木齐市	西北	100	1906	☑
甘肃	甘、陇	兰州市	西北	100	3813	☐
青海	青	西宁市	西北	100	529	☐
陕西	陕、秦	西安市	西北	100	3647	☐
宁夏	宁	银川市	西北	100	527	☑
象牙省	象	象牙市	东华区	200	320	☐
*				0	0	☐

图 8-37　修改记录

8.4.9　处理 SQL 语句中的单引号

如果数据表的记录中本身包含单引号，在书写 SQL 语句时，要把一个单引号替换为连续的两个单引号。

在 SQL 语句中，由于整个语句外侧是两个双引号，因此语句内部一律使用单引号。两个单引号包起来的内容表示常量字符串，如果字符串本身有单引号，则每个单引号需要写成两个单引号才行。

在数据表中，把区域是华北，并且外侧有单引号的记录，相应的省会也加上单引号。也就是说，把石家庄改成'石家庄'，如图 8-38 所示。

根据上述需求，或许会写出如下的 SQL 语句：

```
cnn.Execute “Update Detail Set 省会='''+省会+''',面积=100 Where 区域=''华北'''"
```

Detail

省份名称	简称	省会	区域	面积	人口	少数民族自
黑龙江	黑	哈尔滨	东北	45.48	3813	☐
吉林	吉	长春	东北	18.74	2699	☐
辽宁	辽	沈阳	东北	15.59	4203	☐
北京	京	北京	华北	1.68	1423	☐
天津	津	天津	华北	1.13	1007	☐
河北	冀	石家庄	'华北'		6735	☐
山西	晋	太原	'华北'	15.63	3294	☐
内蒙古	蒙	呼和浩特	华北	118.3	2379	☑
山东	鲁	济南	华东	15.38	9082	☐
江苏	苏	南京	华东	10.26	7381	☐

图 8-38　区域字段包含单引号的记录

但是执行程序时，会出现“操作符丢失”的运行时错误提示，如图 8-39 所示。

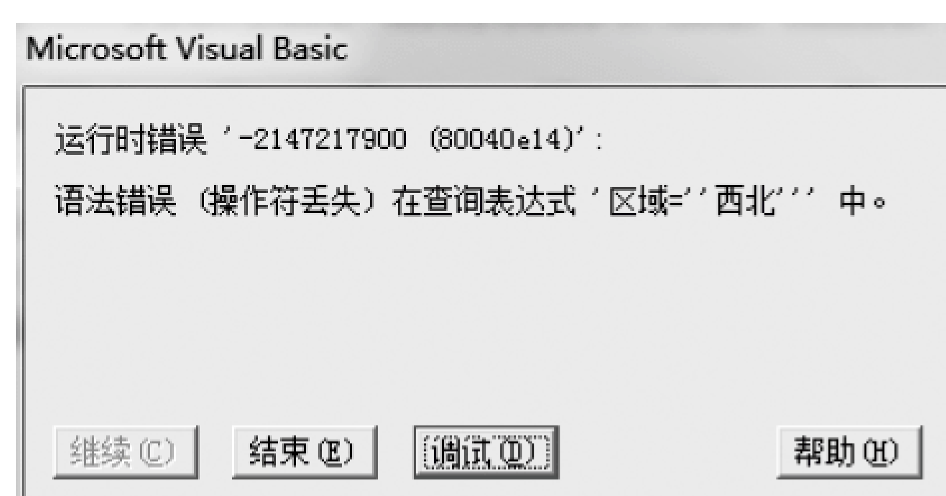


图 8-39　单引号造成的错误

以上语句有两处错误，第一处是省会左右两侧各需要 4 个单引号才行，因为“省会”本身是字段名称，这里要与左右两侧部分进行连接运算，因此 4 个单引号才能表示一个真正的单引号。

另一处错误是 Where 子句中，华北左右两侧各需要 3 个单引号。这是因为华北本身是常量字符串，包含于单引号之中。

修改为如下形式。

```
cnn.Execute “Update Detail Set 省会 ="" & 省会 & "", 面积 =100 Where 区域 ="'华北'""
```

再次执行，可以看到省会的两侧加上了单引号，如图 8-40 所示。

Detail

省份名称	简称	省会	区域	面积	人口	少数民族自
黑龙江	黑	哈尔滨	东北	45.48	3813	☐
吉林	吉	长春	东北	18.74	2699	☐
辽宁	辽	沈阳	东北	15.59	4203	☐
北京	京	北京	华北	1.68	1423	☐
天津	津	天津	华北	1.13	1007	☐
河北	冀	'石家庄'	'华北'	100	6735	☐
山西	晋	'太原'	'华北'	100	3294	☐
内蒙古	蒙	呼和浩特	华北	118.3	2379	☑
山东	鲁	济南	华东	15.38	9082	☐

图 8-40　正确处理单引号

在编程开发过程中，经常要把带有单引号的内容作为查询条件，或者把带有单元格的内容存入数据库。可以利用 VBA 中的 Replace 函数把单引号替换后再执行，这是一个稳妥的做法。

以上内容的源代码文件为“实例文档 46.xlsm”。

8.5　修改数据库结构

SQL 语句除了可以操作现有表的记录外，还可以进行创建数据库，创建和修改数据表，创建和修改字段等操作。

8.5.1　自动创建新数据库

若要自动创建数据库，需要向 VBA 工程添加外部引用“Microsoft ADO Ext 2.8 for DDL and Security”，如图 8-41 所示。

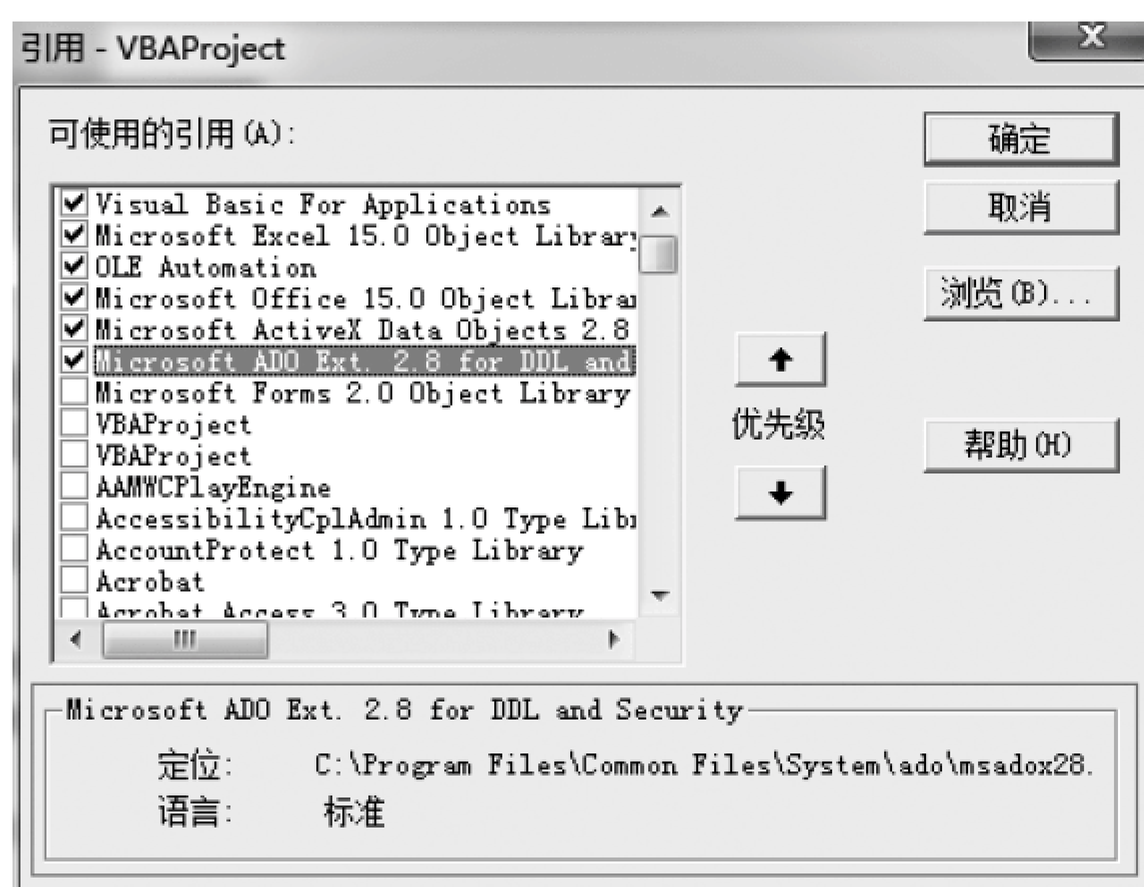


图 8-41　添加外部引用

添加该引用后，运行如下 VBA 过程，即可在工作簿所在路径下自动产生一个 Student.accdb 的新数据库文件。

```
Sub 创建 Access 数据库 ()
    Dim cat As ADOX.Catalog
    Set cat = New ADOX.Catalog
    cat.Create "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & ThisWorkbook.
Path & "\Student.accdb;"
End Sub
```

但是，创建的数据库里面没有任何数据表，是一个空白数据库。

8.5.2　自动创建新表

SQL 语句中的 Create Table 语句用来向数据库中添加新表。

下面的 VBA 过程向已有的 Student 数据库中创建一个“基本信息”表，并且自动添加一条数据记录。

```
Sub 创建数据表 ()
    Dim cnn As ADODB.Connection
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\Student.accdb;Persist Security Info=False;"
        .Open
        .Execute CommandText:="Create Table 基本信息 (姓名 char(4),性别 char(1),
出生日期 Date,年龄 int)"
        .Execute CommandText:="Insert Into 基本信息 Values('刘永富','男','1981-
7-15',37)"
```

```
    End With
    Set cnn = Nothing
End Sub
```

代码分析：“Create Table 基本信息 (姓名 char(4), 性别 char(1), 出生日期 Date, 年龄 int)”这一句是关键代码，意思是创建一个名称为“基本信息”的表，包含姓名、性别、出生日期、年龄四个字段，并且规定了每个字段的类型和字符长度。

运行上述过程，从 Access 中可以看到刚创建的表，如图 8-42 所示。



图 8-42 自动创建新表

并且可以看到该表中已经有一条数据记录。

8.5.3 字段的增加删除和修改

SQL 语句中的 Alert Table 子句可以对现有数据表的结构进行修改。修改字段的语法如下。

- ❑ Add Column 新字段名 类型（长度）：增加新字段。
- ❑ Drop Column 字段名：删除字段。
- ❑ Alter Column 字段名 字段类型（长度）：修改现有字段的类型。

下面的过程向基本信息表增加【住址】和【身份证号码】两个字段，并且删除【出生日期】字段，把【性别】字段的类型从 char 修改为 memo（备注型）。

```
Sub 修改字段 ()
    Dim cnn As ADODB.Connection
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & "\Student.accdb;Persist Security Info=False;"
        .Open
```

```
            .Execute CommandText:="Alter Table 基本信息 Add Column 住址 char(10),身份
证号码 char(18)"
            .Execute CommandText:="Alter Table 基本信息 Drop Column 出生日期"
            .Execute CommandText:="Alter Table 基本信息 Alter Column 性别 memo"
        End With
        Set cnn = Nothing
    End Sub
```

运行上述过程，从 Access 中以设计视图打开“基本信息”表，可以看到各个字段的变化，如图 8-43 所示。



图 8-43　删除和修改字段

8.5.4　自动删除数据表

删除表的 SQL 语句很好写，Drop Table 基本信息，就可以把名为“基本信息”的表从数据库中删除。

以上内容的源代码文件为“实例文档 47.xlsm”。

8.6　访问其他类型的数据库

在编程过程中，要处理的数据未必都存放在 Access 数据库中，很多情况下还存储于 Excel 文件、文本文件中，ADO+SQL 技术，同样适用于数据比较规范的其他数据库。

8.6.1　连接字符串的构造

使用 ADO 能否正常操作和访问一个数据库，关键在于连接字符串是否正确。各种类型数据库的连接字符串如表 8-1 所示。

表 8-1　常见数据库的连接字符串

数据库类型	扩 展 名	连接字符串
Access 2003 数据库	.mdb	Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DBName;Persist Security Info=False;
Access 2007 以上数据库	.accdb	
Access 数据库（有密码）		Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DBName;Jet OLEDB:Database Password=MyDbPassword;
Excel 97-2003 文件	.xls	Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DBName;Extended Properties="Excel 8.0;HDR=YES";

续表

数据库类型	扩 展 名	连接字符串
Excel 2007 以上工作簿	.xlsx	Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DBName;Extended Properties="Excel 12.0 Xml;HDR=YES;IMEX=1";
启用宏的工作簿	.xlsm	Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DBName;Extended Properties="Excel 12.0 Macro;HDR=YES";
文本文件	.txt 或 .csv	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=DBName\;Extended Properties="text;HDR=Yes;FMT=Delimited";

表中提到的 DBName 就是数据库的路径。

针对各种类型数据库的连接方式，可以访问：https://www.connectionstrings.com/excel/、https://www.connectionstrings.com/access/。

8.6.2 查询 Excel 工作表数据

如果一个 Excel 工作簿文件中含有格式规范的数据，也可以在 VBA 中通过 ADO 技术把 Excel 文件当作数据库进行查询。

现在假定磁盘下有一个名为“学生成绩表.xlsm”的 Excel 文件，这个工作簿包含一个名为“第二学期”的工作表，表中有 4 个班级的学生成绩，如图 8-44 所示。

学生成绩表.xlsm - Excel

文件 开始 插入 页面布局 公式 数据 审阅 视图 开发工具 团队

A1 学号

	A	B	C	D	E	F	G	H	I	J	K
1	学号	姓名	性别	班级	数学	语文	物理	化学	英语	体育	总分
2	CY018	金龙宇	男	初一1班	81	58	84	50	70	55	398
3	CY001	姚晨梦	女	初一1班	99	96	90	90	75	88	514
4	CY002	吴虹羽	女	初一1班	88	94	51	57	70	62	422
5	CY003	尤福根	男	初一1班	65	96	53	71	69	80	434
6	CY004	王秋月	男	初一1班	81	82	94	66	87	65	475
7	CY005	赵梦琦	女	初一1班	92	82	97	55	51	63	440
8	CY006	卢佳建	男	初一1班	72	65	64	98	50	81	430
9	CY007	陈梦迪	男	初一1班	98	61	58	85	66	80	448
10	CY008	杨鸿翔	男	初一1班	62	58	82	56	58	96	412
11	CY009	卢雨晴	女	初一1班	83	71	70	77	86	100	487
12	CY010	金磊	男	初一1班	74	98	95	89	85	56	497
13	CY011	周俊杰	男	初一1班	98	74	97	50	62	61	442
14	CY012	吴越	男	初一1班	80	60	62	63	65	82	412
15	CY013	徐思雨	女	初一1班	91	63	88	70	53	90	455
16	CY014	沈玉洁	女	初一1班	54	59	79	71	65	65	393
17	CY015	郭敏燕	女	初一1班	62	90	53	53	66	71	395
18	CY016	杨何波	男	初一1班	80	79	78	80	92	63	472
19	CY017	吴鹏飞	男	初一1班	93	86	54	81	93	60	467
20	CY019	杨晓奇	男	初一1班	100	97	77	92	83	98	547
21	CY020	卢晟	男	初一1班	51	78	83	69	94	52	427
22	CY021	蔡伟豪	男	初一1班	84	97	52	79	98	69	479
23	CY022	罗茜茜	女	初一1班	88	71	96	53	61	59	428
24	CY023	吴一帆	男	初一1班	54	71	51	77	62	81	396
25	CY024	沈璐琦	男	初一1班	90	73	85	52	53	73	426
26	CY025	沈舒婷	女	初一2班	80	95	99	62	73	85	494
27	CY026	杨宇	男	初一2班	97	84	51	77	85	70	464
28	CY027	金彤昕	男	初一2班	56	62	71	92	84	87	452
29	CY028	卢晓雨	男	初一2班	97	90	52	87	91	66	483
30	CY029	叶晴晴	女	初一2班	92	52	73	74	66	94	451
31	CY030	徐世莲	女	初一2班	60	91	64	74	59	64	412

第二学期

图 8-44 Excel 示例数据表

在其他工作簿中创建如下 VBA 过程，用来查询学生成绩表中【班级】为初一 1 班，并

且按【数学】降序排列。

```
Public cnn As ADODB.Connection, rst As ADODB.Recordset
Sub QueryExcel()
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source= " & ThisWorkbook.Path & "\学生成绩表.xlsm;Extended Properties='Excel 12.0 Macro;HDR=YES'"
        .Open
        If .State = ADODB.ObjectStateEnum.adStateOpen Then
            Set rst = New ADODB.Recordset
            rst.CursorLocation = adUseClient
            rst.Open Source:="Select * From [第二学期$] Where 班级='初一1班' Order By 数学 DESC", ActiveConnection:=cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.adLockOptimistic
            If rst.State = ADODB.ObjectStateEnum.adStateOpen Then
                Debug.Print "结果记录集的记录总数：", rst.RecordCount
                Debug.Print "结果记录集的字段总数：", rst.Fields.Count
                For Each fld In rst.Fields
                    Debug.Print fld.Name, fld.Type, fld.Value
                Next fld
                ActiveSheet.Range("A2").CopyFromRecordset rst
                Dim i As Integer
                For i = 0 To rst.Fields.Count - 1
                    ActiveSheet.Cells(1, i + 1).Value = rst.Fields(i).Name
                Next i
                rst.Close                   '关闭 rst
            End If
            .Close                          '关闭 cnn
        Else
            MsgBox "失败！", vbExclamation
        End If
    End With
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

代码分析：与查询 Access 数据库有以下两点不同。

- 连接字符串有所不同，由于数据源是启用宏的工作簿，因此连接字符串后面带有 Extended Properties='Excel 12.0 Macro;HDR=YES'。
- 表名的写法不一样，Access 中的表名直接书写即可，Excel 文件的某个工作表作为数据源，则需要写作：[第二学期$]，如果要把指定的一个区域作为查询的数据源，在 $ 后面加上单元格地址即可，例如：[第二学期$A1:D10]。

运行上述过程，结果记录集打印到活动工作表中，如图 8-45 所示。

可以看到，查询出的记录只有初一 1 班的，而且按数学成绩降序排列。

需要注意的是，查询过程中被查询的工作簿不需要在 Excel 中打开，也就是说 ADO 可以访问处于关闭状态的 Excel 文件。

另外，使用 ADO+SQL 操作 Excel 文件时，支持 Select 语句、Insert Into 语句和 Update 语句，但是不支持 Delete 语句，也就是不能用 Delete 语句删除工作簿中的数据记录。

学号	姓名	性别	班级	数学	语文	物理	化学	英语	体育	总分
CY019	杨晓奇	男	初一1班	100	97	77	92	83	98	547
CY001	姚晨梦	女	初一1班	99	96	90	90	75	88	514
CY011	周俊杰	男	初一1班	98	74	97	50	62	61	442
CY007	陈梦迪	男	初一1班	98	61	58	85	66	80	448
CY017	吴鹏飞	男	初一1班	93	86	54	81	93	60	467
CY005	赵梦琦	女	初一1班	92	82	97	55	51	63	440
CY013	徐思雨	女	初一1班	91	63	88	70	53	90	455
CY024	沈璐琦	男	初一1班	90	73	85	52	53	73	426
CY022	罗茜茜	女	初一1班	88	71	96	53	61	59	428
CY002	吴虹羽	女	初一1班	88	94	51	57	70	62	422
CY021	蔡伟豪	男	初一1班	84	97	52	79	98	69	479
CY009	卢雨晴	女	初一1班	83	71	70	77	86	100	487
CY004	王秋月	男	初一1班	81	82	94	66	87	65	475
CY018	金龙宇	男	初一1班	81	58	84	50	70	55	398
CY012	吴越	男	初一1班	80	60	62	63	65	82	412
CY016	杨何波	男	初一1班	80	79	78	80	92	63	472
CY010	金磊	男	初一1班	74	98	95	89	85	56	497
CY006	卢佳建	男	初一1班	72	65	64	98	50	81	430
CY003	尤福根	男	初一1班	65	96	53	71	69	80	434
CY008	杨鸿翔	男	初一1班	62	58	82	56	58	96	412
CY015	郭敏燕	女	初一1班	62	90	53	53	66	71	395
CY014	沈玉洁	女	初一1班	54	59	79	71	65	65	393
CY023	吴一帆	男	初一1班	54	71	51	77	62	81	396
CY020	卢晟	男	初一1班	51	78	83	69	94	52	427

图 8-45　以 Excel 文件为数据源的查询结果

8.6.3　查询 CSV、TXT 文件

使用逗号分隔值（Comma-Separated Values，CSV，有时也称为字符分隔值，因为分隔字符也可以不是逗号）的文件以纯文本形式存储表格数据（数字和文本）。CSV 文件由任意数目的记录组成，记录间以某种换行符分隔。每条记录由字段组成，字段间的分隔符是逗号或制表符。

Excel 工作表可以另存为 CSV 文件，使用记事本程序也可以编辑数据并另存为 CSV 文件。对于已存在的 CSV 文件，使用 Excel 或记事本程序可以对其进行编辑和修改。

如图 8-46 所示，记录学生成绩的一个 CSV 文件，可以看到该文件包含 5 列（5 个字段），列之间使用逗号作为分隔符。

```
学号,姓名,语文,数学,英语
HG201801,陶阳民,90,78,81
HG201802,黎帅,92,82,91
HG201803,钟松富,88,96,53
HG201804,汤克宏,88,76,58
HG201805,杨兰泉,57,76,76
HG201806,黎书松,71,85,93
HG201807,武平,83,95,95
HG201808,潘青,84,57,86
HG201809,杨英忠,90,54,93
HG201810,贾雄,55,71,68
HG201811,白芬,96,74,93
HG201812,严力,75,74,59
HG201813,韦维风,90,86,94
HG201814,郭辉,54,61,74
HG201815,钱祖兰,94,80,65
HG201816,于超,74,66,54
HG201817,邓阳,83,61,96
HG201818,蒋敬美,60,91,73
HG201819,孟坤,66,90,92
HG201820,阎岩,63,75,91
HG201821,邵平年,69,51,62
HG201822,陶银华,84,91,98
HG201823,田晖,52,85,85
HG201824,萧洁鑫,83,58,97
HG201825,陈珊,75,51,98
```

图 8-46　典型的 CSV 文件

下面讲述如何使用 ADO+SQL 技术查询来自 CSV 文件的数据。

CSV 文件、TXT 文件都属于文本文件，查询文本文件的连接字符串语法格式如下。

```
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=FolderName\;
Extended Properties='text;HDR=Yes;FMT=Delimited';
```

其中，Data Source 指明了文本文件的所在位置。Extended Properties 是扩展属性，其中的 text 表示文本文件，HDR=Yes 时，文本文件的首行被当作字段行；HDR=No 时，认为文本文件没有字段行，所有内容都是记录。

FMT 参数用来规定列与列之间的分隔符。如果是以逗号分隔的 CSV 文件，FMT 设置为 Delimited；如果是以制表位分隔的文本文件，设置为 TabDelimited；如果是以字母 x 分隔的文件，FMT 设置为 Delimited(x)。

需要注意的是，查询文本文件时，SQL 语句中的 From 后面要加上文本文件的名称，例如 Select * From 成绩总表 .csv，如果文本文件的名称本身包含空格，还需要把整个文件名用方括号括起来，例如：

```
Select * From [成绩 总表.csv]
```

如果不加方括号，会引起“From 子句错误”。

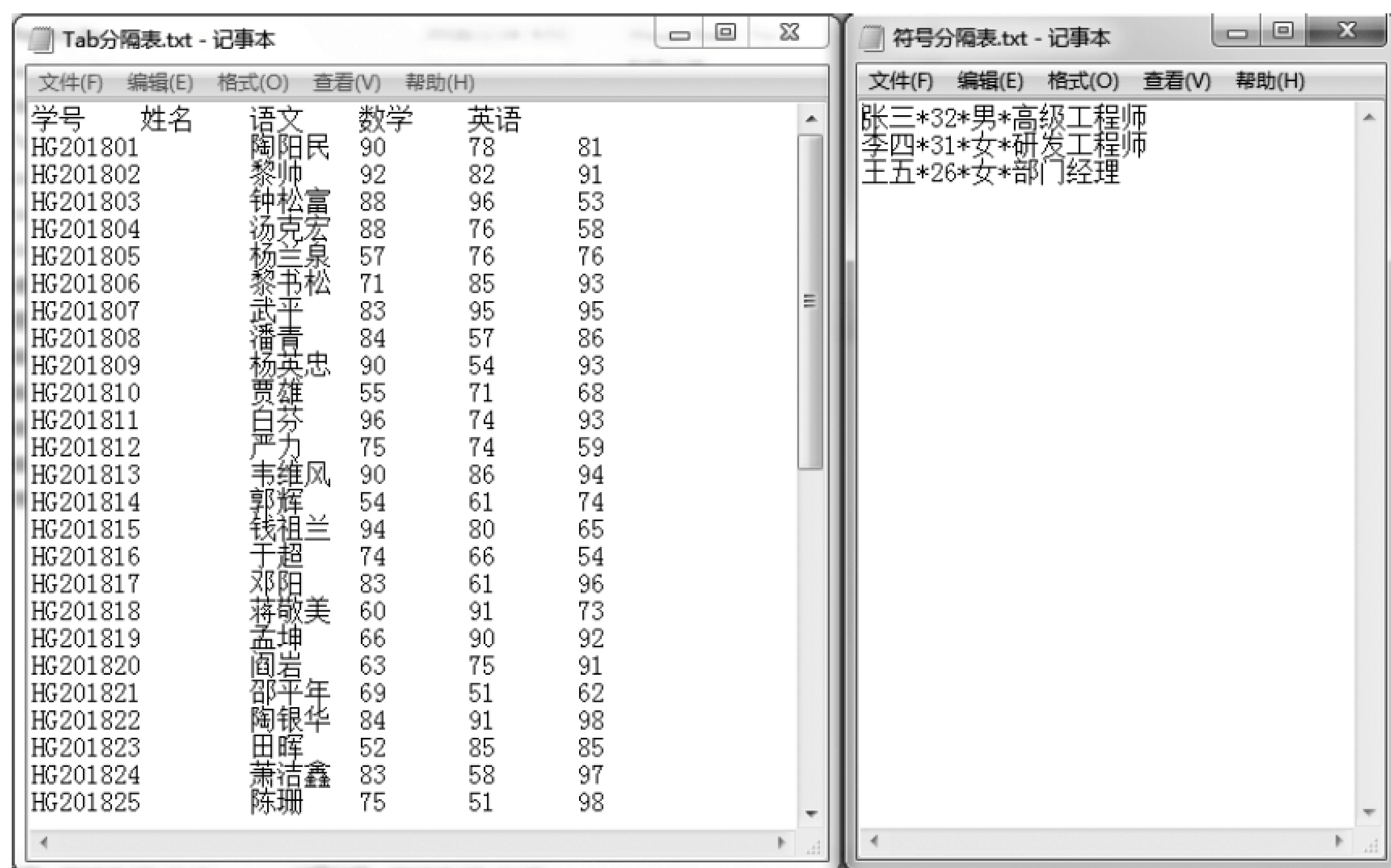
以下程序从“成绩总表 .csv”中查询数学成绩高于 90 分的记录，查询结果发送至工作表中。

```
Sub QueryCSV()
    Set cnn = New ADODB.Connection
    With cnn
        .ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" &
ThisWorkbook.Path & ";Extended Properties='text;HDR=YES;FMT=Delimited'"
        .Open
        Set rst = New ADODB.Recordset
        rst.CursorLocation = adUseClient
        rst.Open Source:="Select * From 成绩总表.csv Where 数学>=90", ActiveConnection:=
cnn, CursorType:=ADODB.CursorTypeEnum.adOpenKeyset, LockType:=ADODB.LockTypeEnum.
adLockOptimistic
        ActiveSheet.Range("A2").CopyFromRecordset rst
        Dim i As Integer
        For i = 0 To rst.Fields.Count - 1
            ActiveSheet.Cells(1, i + 1).Value = rst.Fields(i).Name
        Next i
        rst.Close '关闭rst
        .Close '关闭cnn
    End With
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

上述程序与一般的 ADO+SQL 写法没什么不同，主要关注 ConnectionString 那句代码，还有执行 SQL 查询的那句代码即可。

如果文件夹中存在多种分隔符的文本文件，可以用记事本程序编写一个 schema.ini 文件来表明每个文本文件的分隔符类型、字段名称和数据类型、是否有标题行等属性。

假设文件夹下有一个“ Tab 分隔表 .txt”和一个“符号分隔表 .txt”，其中的内容如图 8-47 所示。



Tab分隔表.txt - 记事本

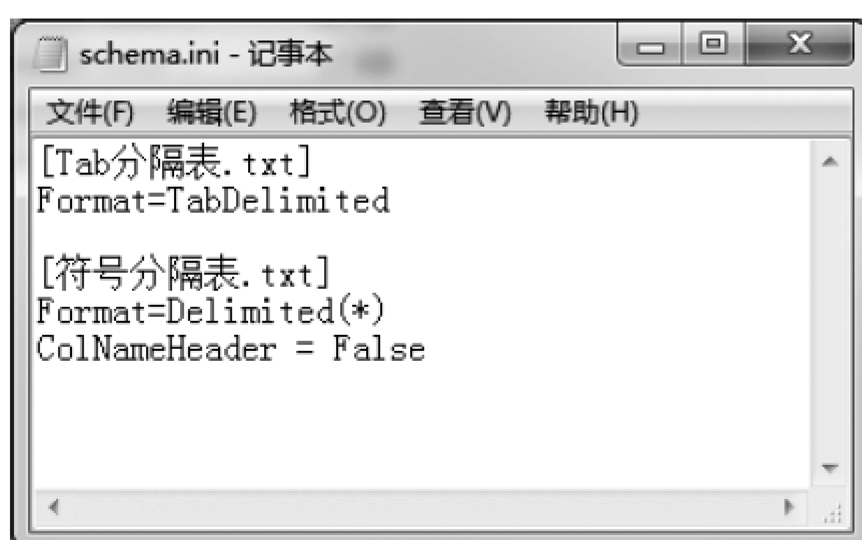
学号	姓名	语文	数学	英语
HG201801	陶阳民	90	78	81
HG201802	黎帅	92	82	91
HG201803	钟松富	88	96	53
HG201804	汤克宏	88	76	58
HG201805	杨三泉	57	76	76
HG201806	黎书松	71	85	93
HG201807	武平	83	95	95
HG201808	潘青	84	57	86
HG201809	杨英忠	90	54	93
HG201810	贾雄	55	71	68
HG201811	白芬	96	74	93
HG201812	严力	75	74	59
HG201813	韦维风	90	86	94
HG201814	郭辉	54	61	74
HG201815	钱祖兰	94	80	65
HG201816	于超	74	66	54
HG201817	邓阳	83	61	96
HG201818	蒋敏美	60	91	73
HG201819	孟坤	66	90	92
HG201820	阎岩	63	75	91
HG201821	邵平年	69	51	62
HG201822	陶银华	84	91	98
HG201823	田晖	52	85	85
HG201824	萧洁鑫	83	58	97
HG201825	陈珊	75	51	98

符号分隔表.txt - 记事本

张三*32*男*高级工程师
李四*31*女*研发工程师
王五*26*女*部门经理

图 8-47　不同分隔符的 CSV 文件

然后创建一个记事本文件，书写如下内容，并另存为 schema.ini，如图 8-48 所示。



schema.ini - 记事本

```
[Tab分隔表.txt]
Format=TabDelimited

[符号分隔表.txt]
Format=Delimited(*)
ColNameHeader = False
```

图 8-48　schema.ini 文件的内容

文件内容的含义是，“ Tab 分隔表 .txt”这个文件的分隔符是 Tab 制表位，“符号分隔表 .txt”这个文件的分隔符是 *，而且这个文件没有标题行。

即使程序代码中仍然使用前面讲过的连接字符串：

```
ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & ThisWorkbook.Path & ";Extended Properties='text;HDR=YES;FMT=Delimited'"
```

由于文本文件路径下存在这个 schema.ini 文件，因此以这个文件中的设定为准。

其中，根据“符号分隔表 .txt”这个文件查询的结果，如图 8-49 所示。

由于该文件没有标题行，所以返回的结果记录集的字段名称使用默认的 F1、F2....。

如果文件夹下没有 schema.ini 文件，运行同样的程序不能得到期望的效果，往往不能正常分列，如图 8-50 所示。

	A	B	C	D
1	F1	F2	F3	F4
2	张三	32	男	高级工程师
3	李四	31	女	研发工程师
4	王五	26	女	部门经理
5				
6				
7				

图 8-49　基于 schema 文件的查询结果

	A	B
1	张三*32*男*高级工程师	
2	李四*31*女*研发工程师	
3	王五*26*女*部门经理	
4		
5		

图 8-50　不指定 schema 文件的查询结果

8.6.4　文本文件的快速合并

当一个文件夹中有大量文件格式相同、内容不同的 CSV 或 TXT 文件，很多情况下需要合并成一个总体文件。自然可以使用前面讲过的查询 CSV 文件的方法，逐一查询每个文件，把结果记录集发送到同一个 Excel 表中。

这里介绍一种利用 Shell 调用 DOS 命令实现快速合并文件的方法，如图 8-51 所示。通常情况下，在命令提示符窗口中输入 copy *.csv Total.csv 可以把当前路径下所有的 CSV 文件合并为 Total.csv。

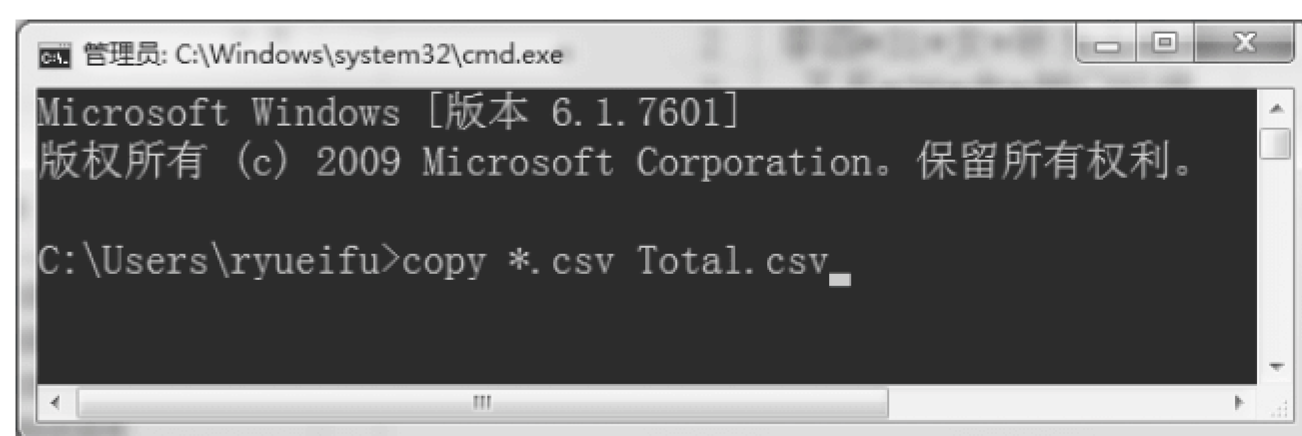


图 8-51　使用 DOS 命令快速合并 csv 文件

使用 VBA 调用上述功能，需要事先把当前路径切换至待处理的 CSV 文件所在路径，然后用 Shell 语句即可。

假设一个文件夹下有 4 个单独的 CSV 文件，运行下面的程序可以合并为一个文件。

```
Sub MergeFiles()
    Dim Path As String
    Path = ThisWorkbook.Path & "\CSV"
    ChDrive Left(Path, 2)
    ChDir Path
    Shell "cmd /c copy Score*.csv summary.csv", vbHide
    Application.Wait Now + TimeValue("00:00:01")
End Sub
```

代码分析：VBA 中的 ChDir 用于更改当前目录，但是必须先用 ChDrive 切换到相应的磁盘分区根目录才行。

另外，Shell 命令是异步执行的，因此最好在 Shell 语句之后加上必要的延时。

运行上述程序，可以看到文件夹中多了一个 summary.csv，用记事本程序查看其内容，不仅写入了每个 CSV 文件的数据记录，而且复制了标题行，如图 8-52 所示。

如果要查询上述总体文件中所有的成绩记录，可以执行如下 SQL 查询命令。

```
Select * From summary.csv Where 学号 Like 'HG%'
```

上面讲过的使用 copy 命令合并文件的方法同样适用于扩展名为 .txt 的文本文件。

以上内容的源代码文件为“实例文档 48.xlsm”。

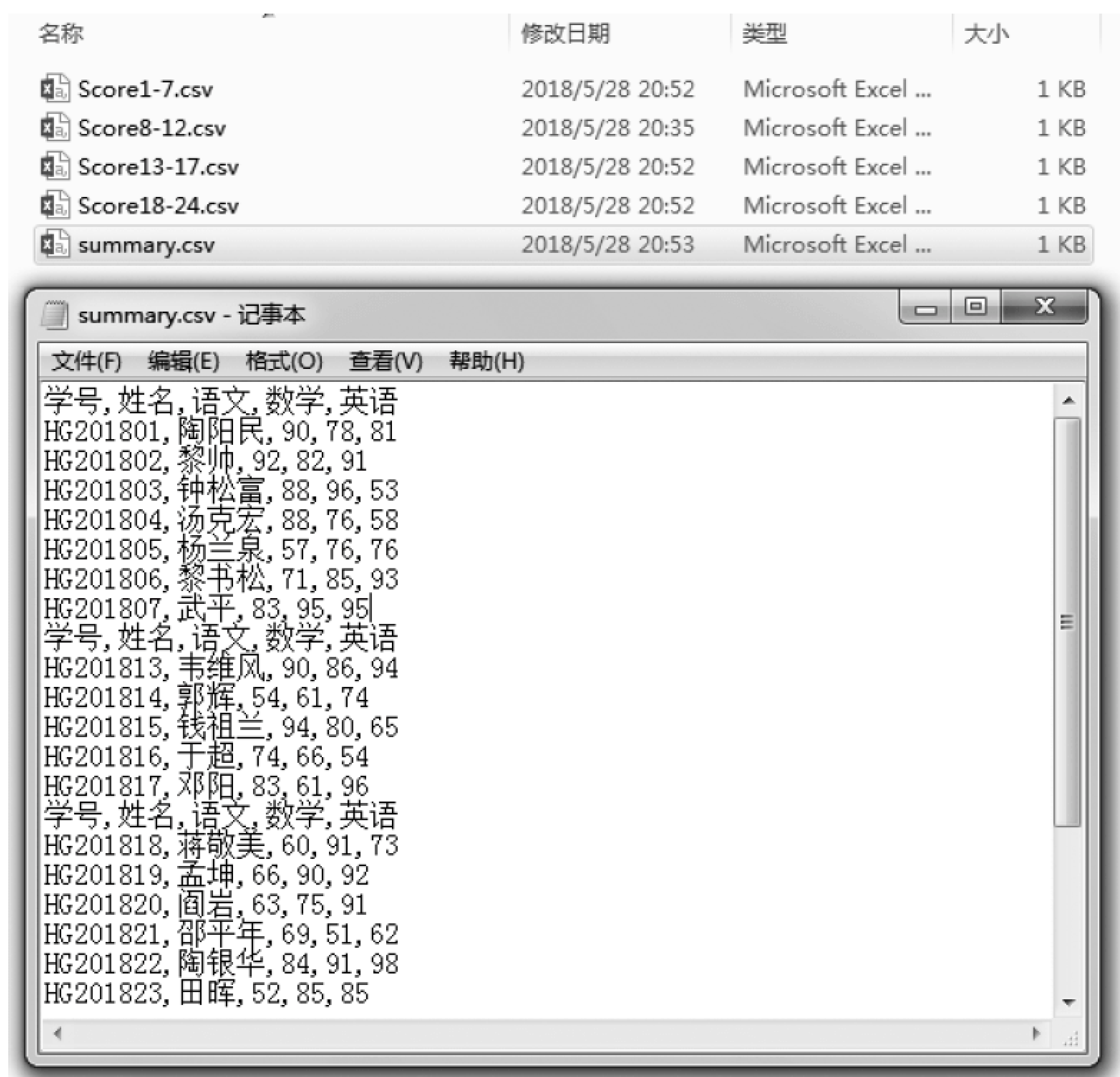


图 8-52　自动合并多个 CSV 文件

8.7　本章小结

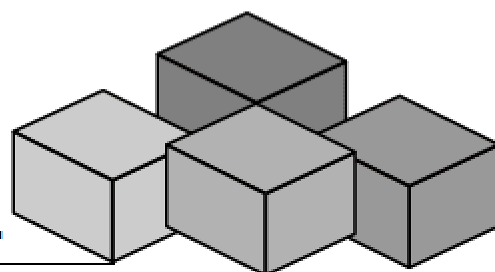
数据表由字段和记录构成，字段就是列名、表头，字段对象（Field）的重要属性有 Name、Type 和 Value。Name 是字段的名字，Type 是字段的数据类型，Value 是当前记录行对应的值。

VBA 中引用 ADODB 对象可以访问 Access、Excel 工作表、文本文件等数据库，查询的结果是一个 ADODB.RecordSet 对象，使用该对象的 MoveNext 等方法实现记录行的跳转，BOF 属性用于判断游标是否处于第一行记录之前，EOF 属性判断游标是否处于最后一行记录之后。

常用的 SQL 语句有 Insert Into、Delete、Update 和 Select。

第 9 章

Office VBA 混合编程



微软 Office 的很多组件，与 Excel VBA 一样，都支持 VBA 编程。一般情况下每个组件的 VBA 编程所操作的是对应组件的对象模型，微软公司提供的 OLE 自动化技术，可以通过一个应用程序来控制另外应用程序，例如可以在 Excel VBA 中对 Word 的对象进行读写。

Office VBA 的跨组件编程使得不同组件的数据可以在同一个程序中共享使用。同时，跨组件编程也是使用其他语言进行 Office 开发的基础，具体关系如图 9-1 所示。

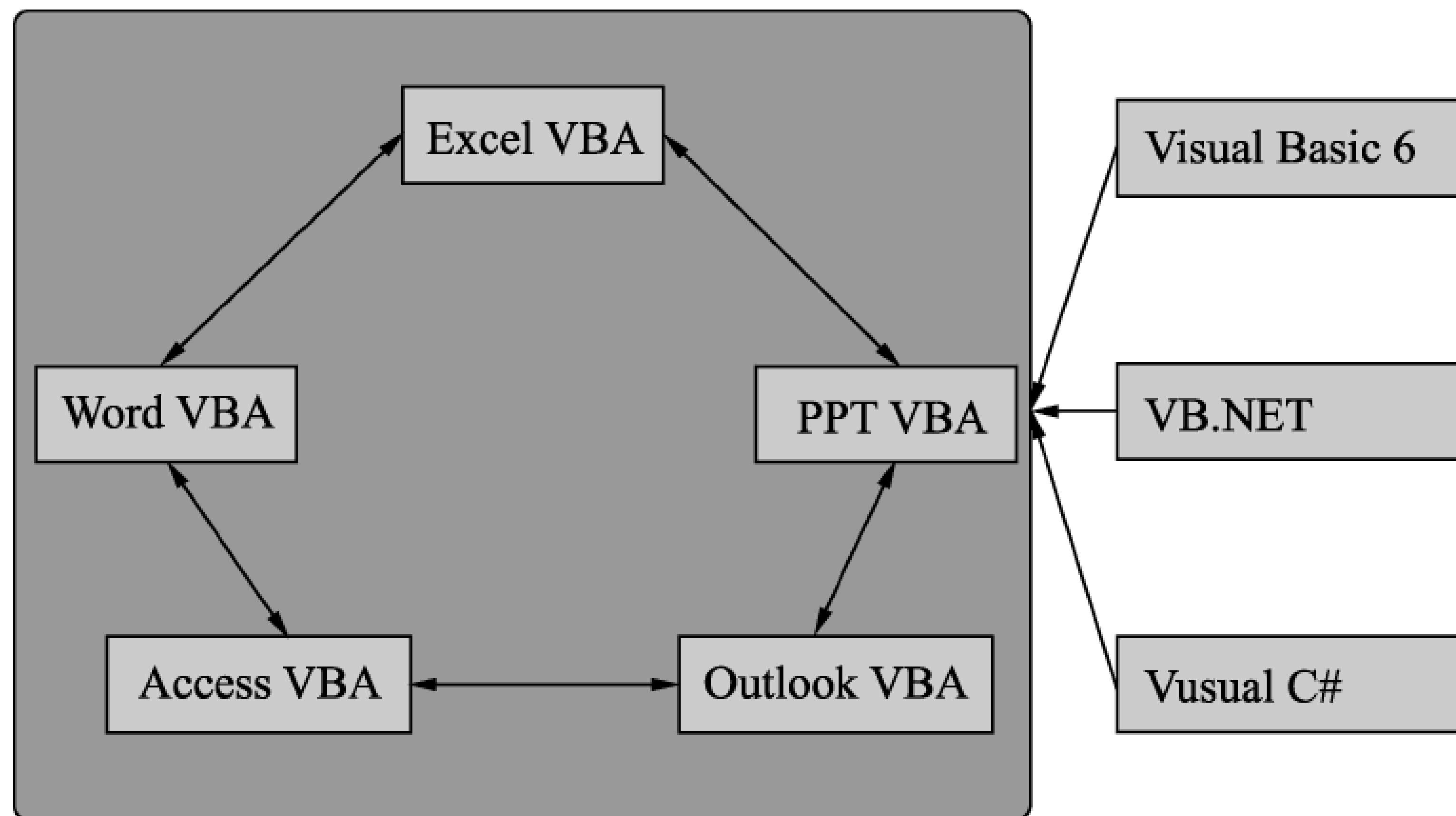


图 9-1　多种语言可以访问 Office 对象

本章首先介绍前期绑定和后期绑定，然后介绍应用程序对象的获取和创建，最后讲述几个不同组件互相访问的实例。

9.1　前期绑定和后期绑定

严格地讲，操作其他组件与绑定与否没什么关系。所谓的前期绑定，就是在写程序之前，把被控组件的对象模型引入主控程序的工程中。

假设在 Excel VBA 中访问 Word 对象，那么主控程序就是 Excel，被控组件是 Word。此

时的前期绑定就是向 Excel VBA 的工程中添加 Word 对象库。

在 Excel VBA 的代码窗口中，单击【工具 / 引用】，勾选"Microsoft Word 15.0 Object Library"，然后单击"确定"按钮，如图 9-2 所示。

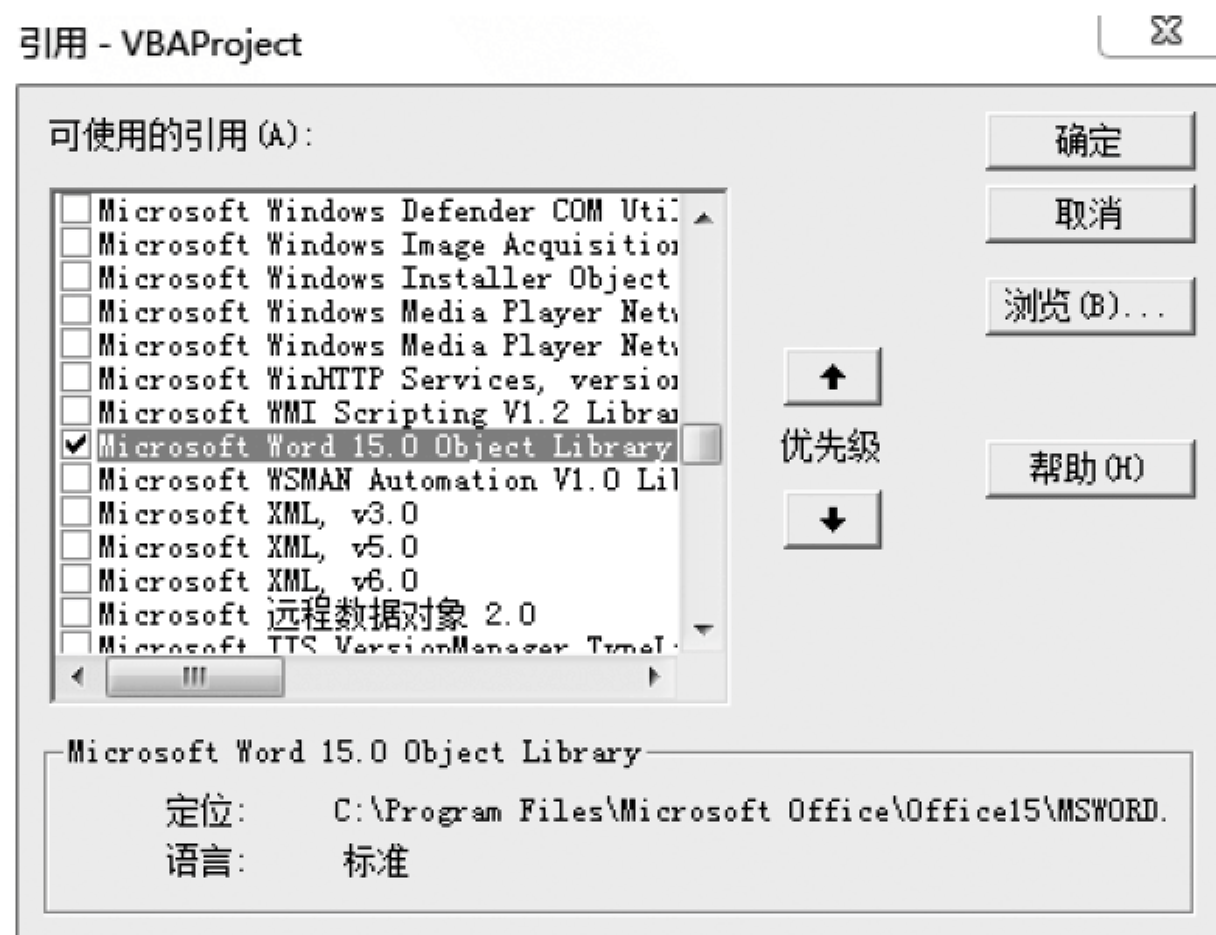


图 9-2 Excel VBA 工程中添加 Word 对象的引用

9.1.1 绑定前后的变化

添加了被控组件的对象库后，书写代码时，可以声明被控组件的对象变量类型，也可以在程序中使用被控组件的枚举常量。

向 Excel VBA 添加 Word 对象库后，可以声明 Word 方面的对象类型，也可以使用 Word 对象库中的枚举常量，如图 9-3 所示。

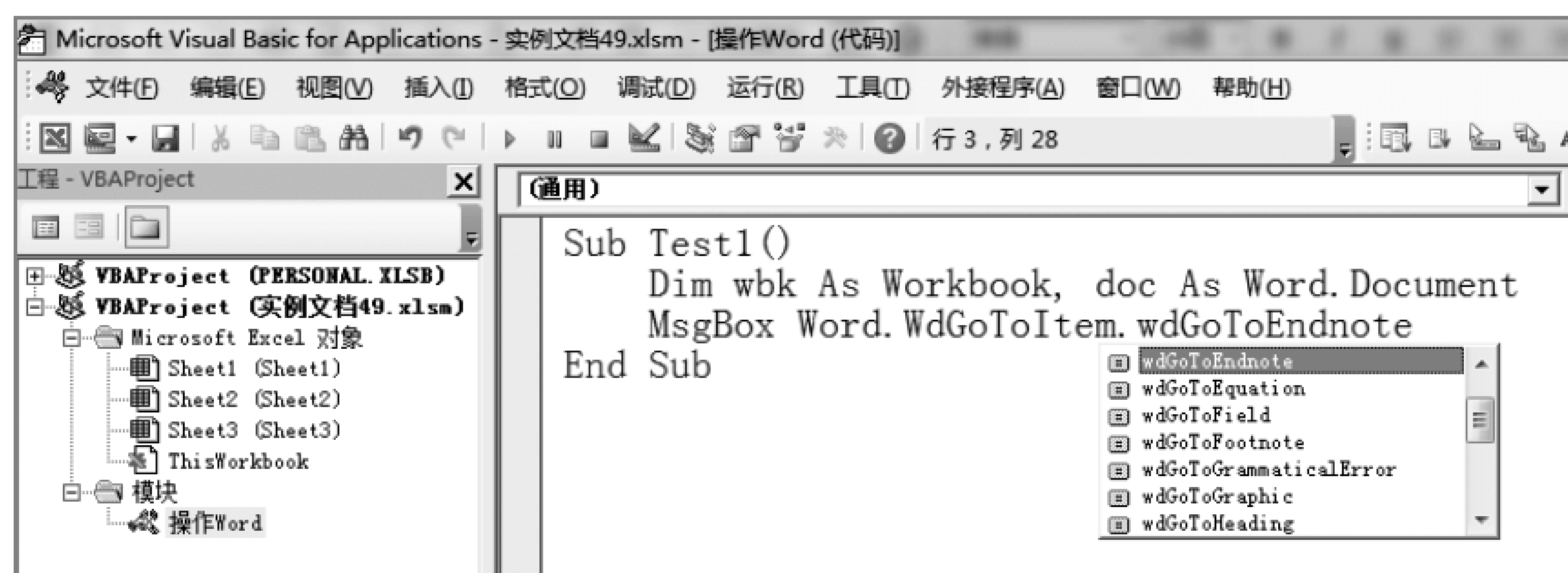


图 9-3 Excel VBA 中可以使用 Word 的枚举常量

例如，运行上述 Test1 过程，弹出的对话框中显示 5（wdGoToEndnote 等于 5）。

被控对象库添加进来以后，对象浏览器中会看到有新增的对象。在 Excel VBA 中按下 F2 键，组合框中选择"Word"和"Document"并按回车键，就可以看到被控组件的对象模型和枚举常量，如图 9-4 所示。

现在取消勾选 VBA 工程中的 Word 引用，再次运行上述过程，将出现"编译错误"，如图 9-5 所示。

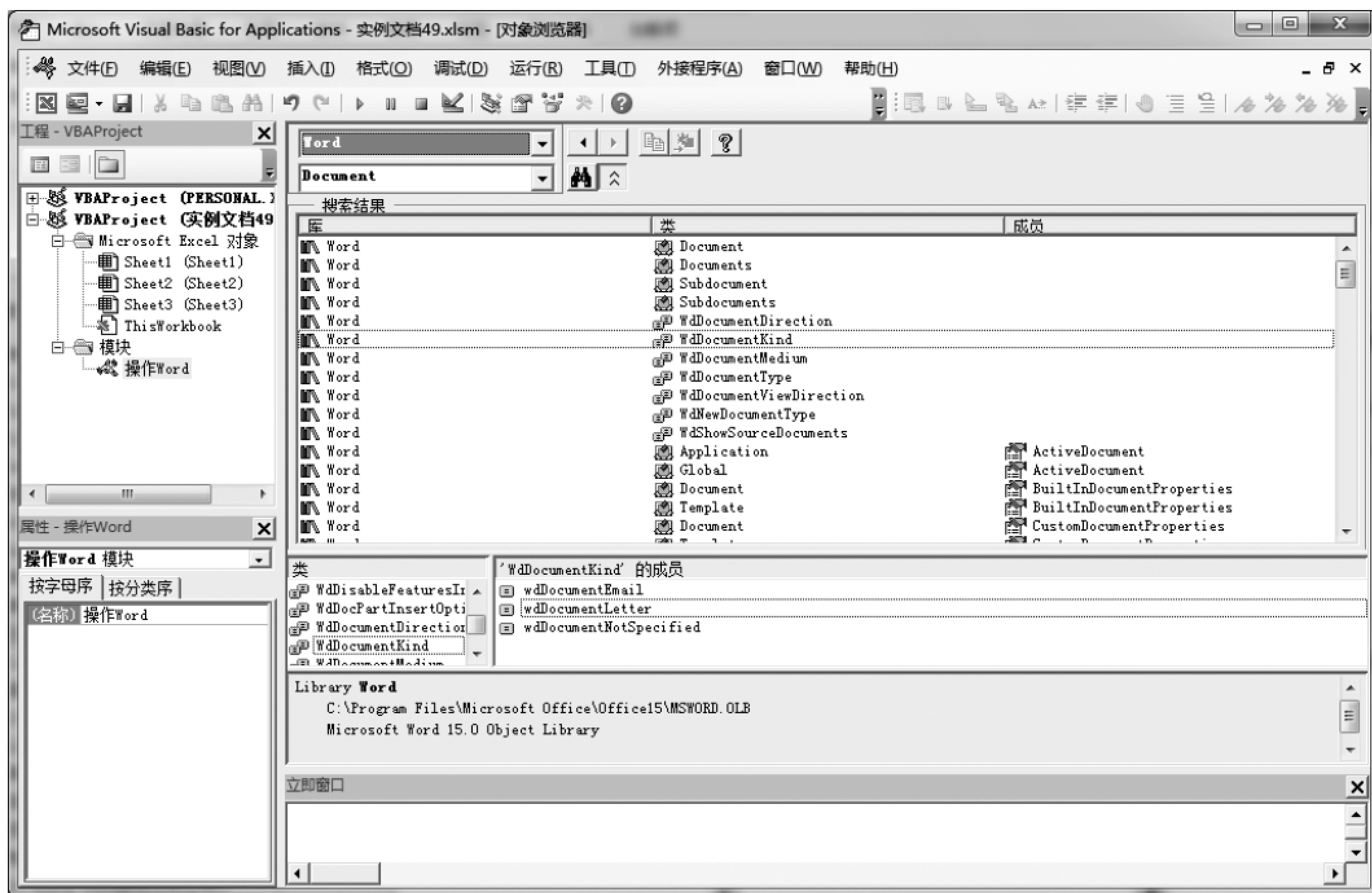


图 9-4　在对象浏览器中查看外部引用库中的成员



图 9-5　不进行前期绑定就不能使用对象库

可以看出，不使用前期绑定，就无法使用被控组件的对象类型和枚举常量。

9.1.2　后期绑定方式

后期绑定方式，指的是不添加被控组件的对象库，对象变量需要声明为 Object 类型，枚举常量使用对应的整型值。

例如，在 Excel VBA 中，下面的程序的作用是获取 Word 应用程序，自动创建一个文档，并且写入内容。

```
Sub Test1()
    Dim wordApp As Object, doc As Object
    Set wordApp = GetObject(, "Word.Application")
    Set doc = wordApp.Documents.Add()
```

```
    doc.Content.InsertAfter Text:="Hello,Excel VBA!"
End Sub
```

由于后期绑定方式没有把被控对象引入工程中，因此存在诸多不便。

9.2 创建和获取应用程序对象

如果被控应用程序还没有运行，可以通过主控程序创建被控应用程序的对象。被控程序的应用程序对象也是被控程序的顶级对象，例如 Word、PowerPoint、Outlook 等也都有相应的 Application 对象。

其实不管是哪一个组件的 VBA，只有通过 Application 对象才能访问到其他对象，因此跨 Office 组件 VBA 编程必须先创建或获取被控组件的应用程序对象，才能进一步访问。

CreateObject 用于创建一个新的应用程序对象，GetObject 用于获取已经在运行的应用程序对象。

9.2.1 使用 CreateObject

下面的程序使用 CreateObject 创建一个新的 Word 应用程序。

```
Sub Test2()
    Dim WordApp As Object
    Set WordApp = CreateObject(Class:="Word.Application")
    With WordApp
        .Visible = True
        .Documents.Add
    End With
End Sub
```

代码分析：CreateObject(Class:="Word.Application") 用来创建一个新对象，“Word.Application”是存储于注册表中的类名，如果计算机中没有安装 Word 组件，那么该代码会创建失败。

运行上述 Excel VBA 过程，可以看到桌面上自动启动 Word 应用程序，并且自动创建了一个空白文档。

9.2.2 使用 New 关键字

使用 New 来创建新应用程序对象时，必须采用前期绑定方式。下面的过程自动创建一个新的 Word 应用程序，并且设置该应用程序可见，窗体状态为最大化。

```
Sub Test3()
    Dim WordApp As Word.Application
    Set WordApp = New Word.Application
    With WordApp
        .Visible = True
        .WindowState = Word.WdWindowState.wdWindowStateMaximize
```

```
    End With
End Sub
```

9.2.3　获取正在运行的应用程序对象

与 CreateObject 相对应的是 GetObject，该函数可以获取已经运行的应用程序，也就是并不创建新对象。

下面的过程获取正在运行的 Word，并打印出当前 Word 打开的文档个数。

```
Sub Test4()
    Dim WordApp As Word.Application
    Set WordApp = GetObject(, Class:="Word.Application")
    With WordApp
        Debug.Print "打开的文档个数：", .Documents.Count
    End With
End Sub
```

代码分析：需要注意的是，GetObject 括号内有一个半角逗号。

如果被控对象并未运行，则会出现“运行时错误 429”提示。

例如，下面的 Excel VBA 代码尝试获取正在运行的 PowerPoint。

```
Sub Test5()
    Dim PowerPointApp As PowerPoint.Application
    Set PowerPointApp = GetObject(, Class:="PowerPoint.Application")
    With PowerPointApp
        .WindowState = PowerPoint.PpWindowState.ppWindowMaximized
    End With
End Sub
```

由于事先没有打开 PowerPoint，所以运行到 GetObject 那行时弹出错误，如图 9-6 所示。

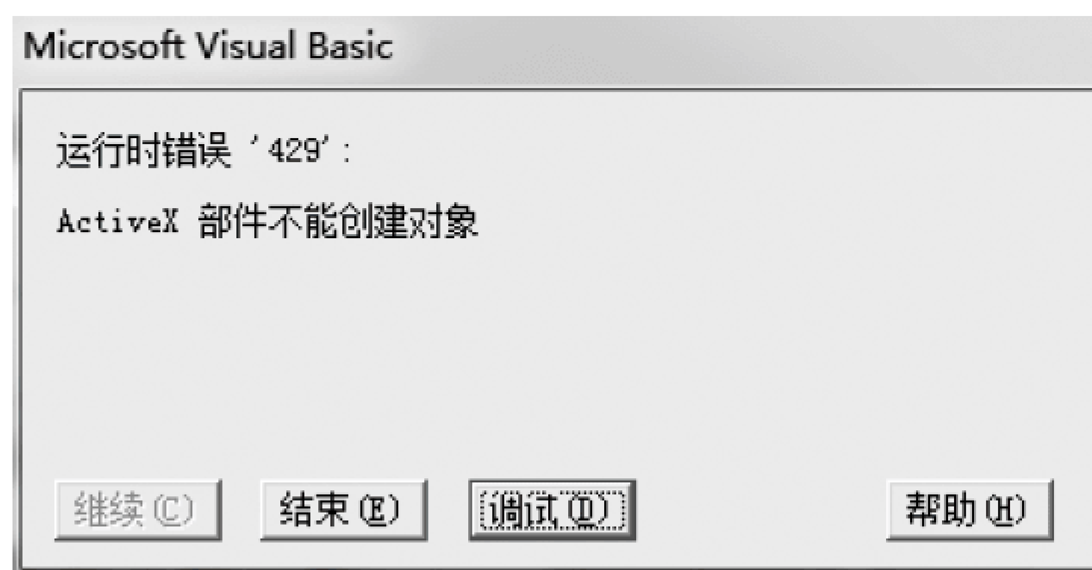


图 9-6　获取不到相应的应用程序

如果已经打开 PowerPoint，运行上述过程则没问题。

GetObject 除了可以获取应用程序外，还可以用于获取磁盘中未打开的文档。运行下面的过程时，在 Excel 中并不会看到文档被打开，但是这个文档的 B1 单元格会被修改。

```
Sub 后台打开Excel文件()
    Dim wbk As Excel.Workbook
    Set wbk = GetObject("C:\temp\333.xlsx")
    wbk.Worksheets("Sheet1").Range("B1").Value = "Happy"
```

```
    wbk.Save
    wbk.Close
    Set wbk = Nothing
End Sub
```

当然也适用于其他 Office 组件，下面的过程后台打开 Word 文档，并且删除文档中的第 2 段落，然后保存并关闭文档。

```
Sub 后台打 Word 文件 ()
    Dim doc As Word.Document
    Set doc = GetObject("C:\temp\ 新员工入职须知 .doc")
    doc.Paragraphs(2).Range.Delete
    doc.Save
    doc.Close
    Set doc = Nothing
End Sub
```

9.3 代码改写技巧

跨组件的 VBA 编程，重点和难点是现有代码的移植和重写，特别是以前一直使用单组件编程，会养成很多简写的习惯，那么在使用跨组件编程时就需要改掉这些习惯。

那么如何把单组件的 VBA 代码修改成可以在其他组件中运行的代码呢？需要注意的有以下 5 点。

- 声明被控组件的对象变量时，使用完全路径，而不是简写形式。
- 使用被控组件的枚举常量时，使用完全路径。
- 被控组件的应用程序对象不能直接使用 Application，而是使用 CreateObject 或 GetObject 来获得。
- 被控组件中用到的对象尽量从被控组件的 Application 对象写起，而不是简写形式。
- 被控组件用到的 CodeName，例如 ThisWorkbook、Sheet1 等，在主控程序中不识别，需要改写为其他形式。

9.3.1 Word VBA 中运行 Excel VBA 代码

假设 Excel 文件中有如下一个 VBA 过程。

```
Public Sub Test6()
    Dim wbk As Workbook, wst As Worksheet
    Set wbk = ThisWorkbook
    Application.DisplayAlerts = False
    Sheet3.Delete
    Set wst = wbk.Worksheets.Add(Before:=Sheet2, Type:=xlWorksheet)
    wst.Name = Format(Date, "YYYYMMDD")
    wst.Range("B2:D5").Select
    Selection.Interior.Color = vbBlue
End Sub
```

上述代码是 Excel VBA 中非常普通的一个宏，功能是首先删除 CodeName 为 Sheet3 的工作表，然后向 Sheet2 之前添加一个新工作表，并赋给变量 wst，重命名工作表为当前日期，最后选择一个单元格区域，涂上背景色。

如果把上述过程原封不动复制到 Word VBA 中，还能够正常运行吗？

下面启动 Word 2013，新建一个文档，另存为“实例文档 49.docm”，打开 Word VBA 编程界面，然后插入一个标准模块。

由于要从 Word VBA 访问 Excel 对象，因此需要添加对 Excel 的外部引用，如图 9-7 所示。

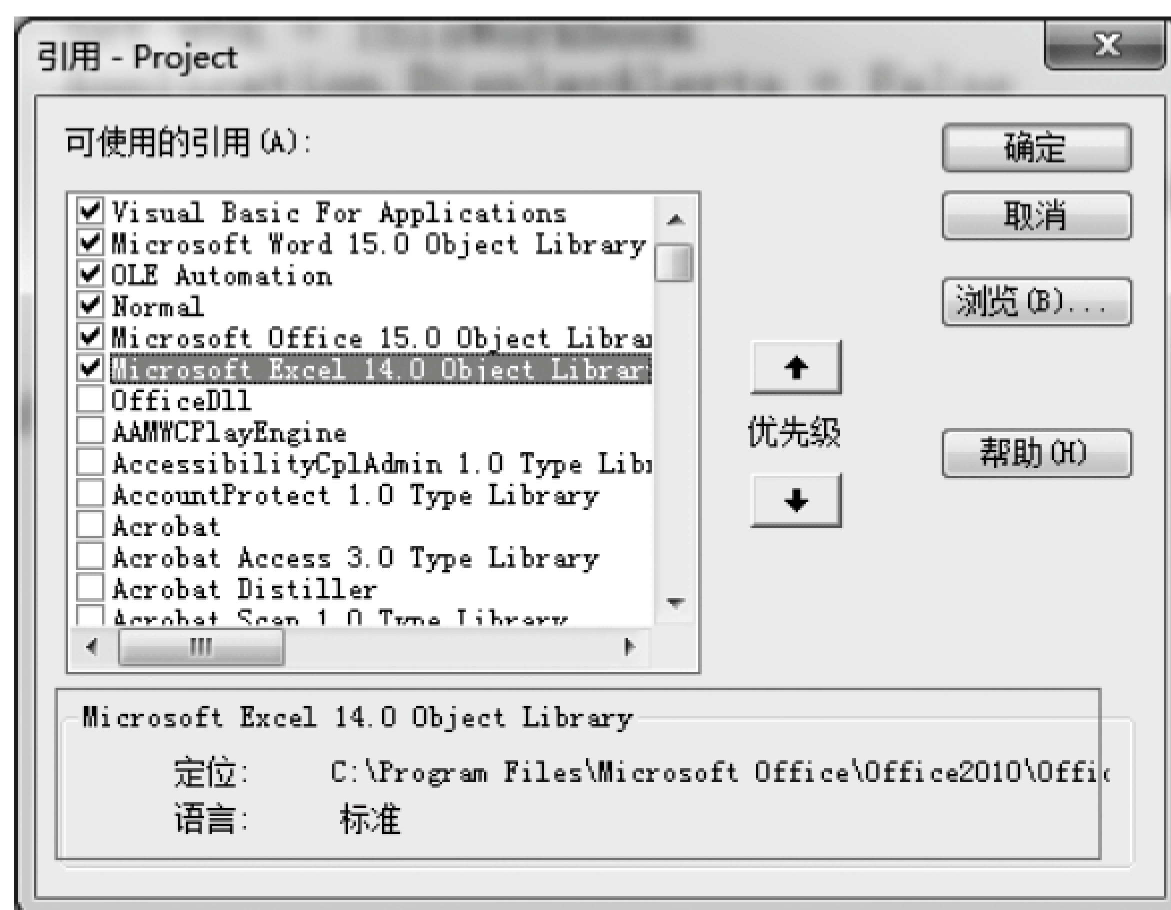


图 9-7　Word VBA 中添加 Excel 的引用

然后把 Excel VBA 中的 Test6 过程复制到 Word VBA 的标准模块中运行。运行肯定不成功！因为需要改写。

按照前面讲过的 5 条改写原则，修改为如下的 Word VBA 过程。

```
Public Sub Test6()
    Dim ExcelApp As Excel.Application
    Dim wbk As Excel.Workbook, wst As Excel.Worksheet
    Set ExcelApp = GetObject(, "Excel.Application")
    Set wbk = ExcelApp.ActiveWorkbook
    ExcelApp.DisplayAlerts = False
    wbk.Worksheets(3).Delete
    Set wst = wbk.Worksheets.Add(Before:=wbk.Worksheets(2), Type:=Excel.XlSheet
Type.xlWorksheet)
    wst.Name = Format(Date, "YYYYMMDD")
    wst.Range("B4:D3").Select
    ExcelApp.Selection.Interior.Color = vbRed
End Sub
```

代码分析：由于 Excel 此时变成了被控程序，因此需要创建一个 ExcelApp 应用程序对象，而不是直接使用 Application，此时的 Application 是主控程序 Word。

原先的 ThisWorkbook 需要改写为 ExcelApp.ActiveWorkbook。

原先的 Type:= xlWorksheet 需要改写为 Type:=Excel.XlSheetType.xlWorksheet，也就是尽

量从被控程序的对象库名称开始写，而不是简写。

在最后一行代码中，原先的 Selection 需要在前面添加 ExcelApp.。因为 Selection 在此处有歧义，Word 也有 Selection 对象。

再次运行修改后的 Test6 过程，可以发现 Word VBA 成功修改了 Excel 这边的内容。

9.3.2 处理被控组件的事件过程

Excel VBA 的很多对象都支持事件编程，例如 Excel VBA 中的 Application、Workbook、Worksheet 都支持很多事件。

假设工作表 Sheet1 后面创建有如下 SelectionChange 事件。

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Target.Interior.Color = vbGreen
    Application.StatusBar = Target.Address
End Sub
```

那么当鼠标选中单元格区域时，该区域会变绿，同时可以看到 Excel 的状态栏里显示了所选区域的地址。

那么以上这个事件过程能不能改写到其他 Office 组件中，也就是让其他组件能够感知 Excel 这边的动作呢?

下面讲解一下，在 PowerPoint VBA 创建 Excel 的工作表单元格区域的选择事件。

启动 PowerPoint 2013，新建一个演示文稿，另存为“实例文档 50.pptm”。打开 PowerPoint 的 VBA 编辑器，单击【工具 / 引用】，添加对 Excel 的外部引用。

然后向该演示文稿的 VBA 工程添加一个类模块，重命名为 ExcelEvent，类模块中的代码如下。

```
Public WithEvents WST As Excel.Worksheet

Private Sub WST_SelectionChange(ByVal Target As Excel.Range)
    Target.Interior.Color = vbRed
     Application.ActivePresentation.Slides(1).Shapes(1).TextFrame.TextRange.
Text = Target.Address(False, False)
End Sub
```

代码分析：WST 是一个带有事件的对象，当选中区域后，区域会变红，而且第 1 张幻灯片的第一个文本框会显示所选 Excel 单元格的相对地址。

接下来需要对上述类进行实例化。

向 PowerPoint VBA 再添加一个标准模块，重命名为 Module1，然后书写如下代码。

```
Public Instance As New ExcelEvent
Public ExcelApp As Excel.Application

Public Sub Main()
    Set ExcelApp = GetObject(, "Excel.Application")
```

```
    Set Instance.WST = ExcelApp.ActiveSheet
End Sub
```

代码分析：Instance 是对 ExcelEvent 的一个实例，需要在 Main 过程中把 Instance 的 WST 与运行中的 Excel 当前工作表进行关联。

代码写好后，手动运行 Main 过程，选择 Excel 单元格区域 B3:C4 后，该区域背景色变红，同时可以看到 PowerPoint 幻灯片中的内容发生相应改变，如图 9-8 所示。

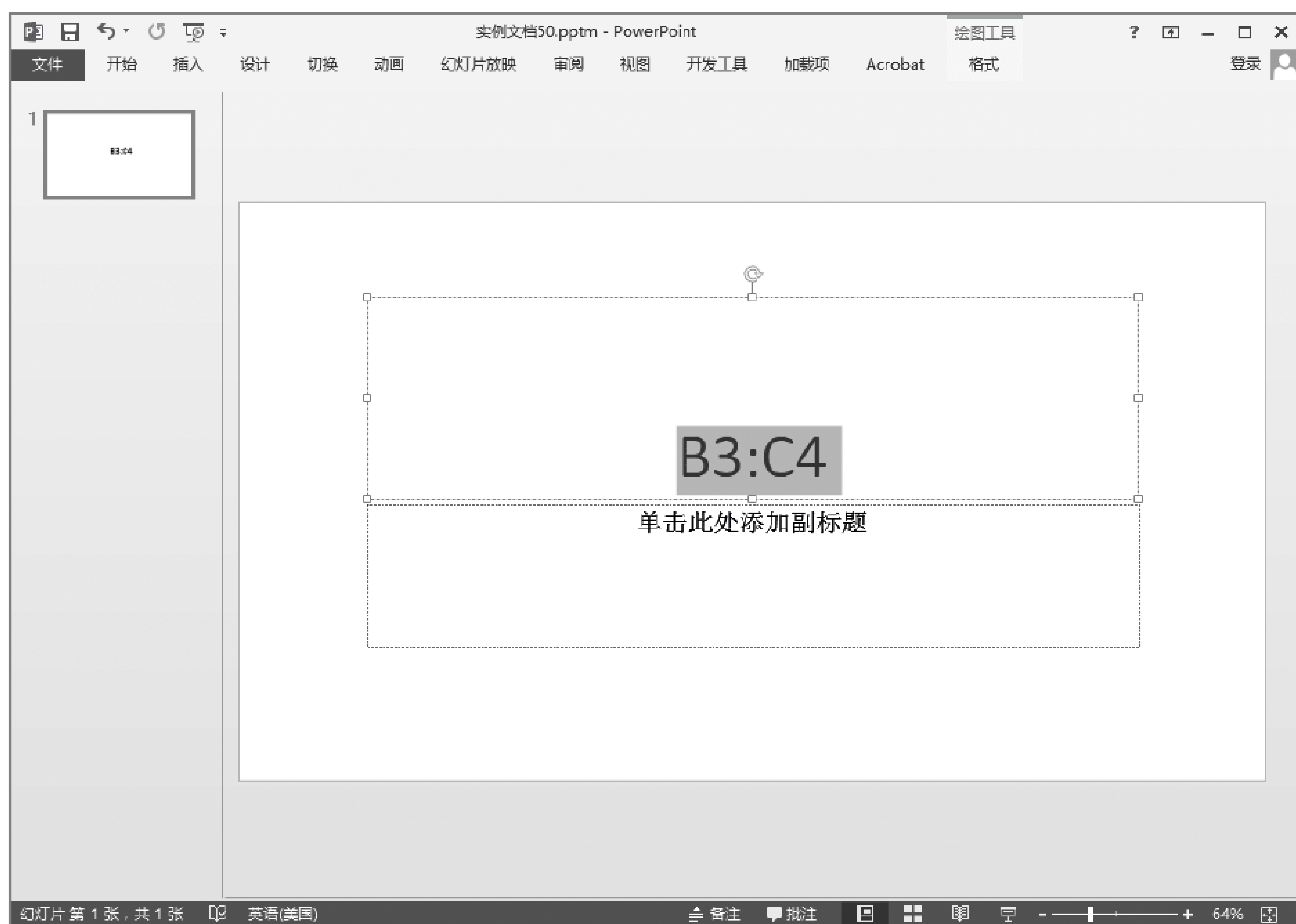


图 9-8　选择 Excel单元格导致 PPT 文本框内容变化

以上实例演示了 Worksheet 对象的事件过程的移植，对于 Application、Workbook 的事件，也可以用上述方法实现，读者自行尝试。

VBA 代码移植到其他组件就实现了 Office VBA 的混合编程，如果把 VBA 代码移植到 VB 或其他语言中，代码被真正地保护了起来，就是 VBA 的封装技术了。关于封装技术，本书暂不讨论。

9.4　跨组件编程实例

Office 跨组件编程，能够把不同组件的数据、对象整合到一个程序中，多个 Office 组件之间实现数据共享，因而意义非常重大。然而跨组件编程的水平高低，仍然取决于单组件的 VBA 对象模型的理解程度。

9.4.1　Word VBA 调用 Excel 工作表函数实现英汉互译

Excel 2013 的新增工作表函数可以进行中英文互译，现在 Word 文档中有一些单词需要

翻译，这就需要借用 Excel 的功能来完成。

“实例文档 51.docm”这个 Word 文档包含一个表格，表格的第 1 列是一些词汇，第 2 列是空白的，准备把翻译的结果放入第 2 列，如图 9-9 所示。

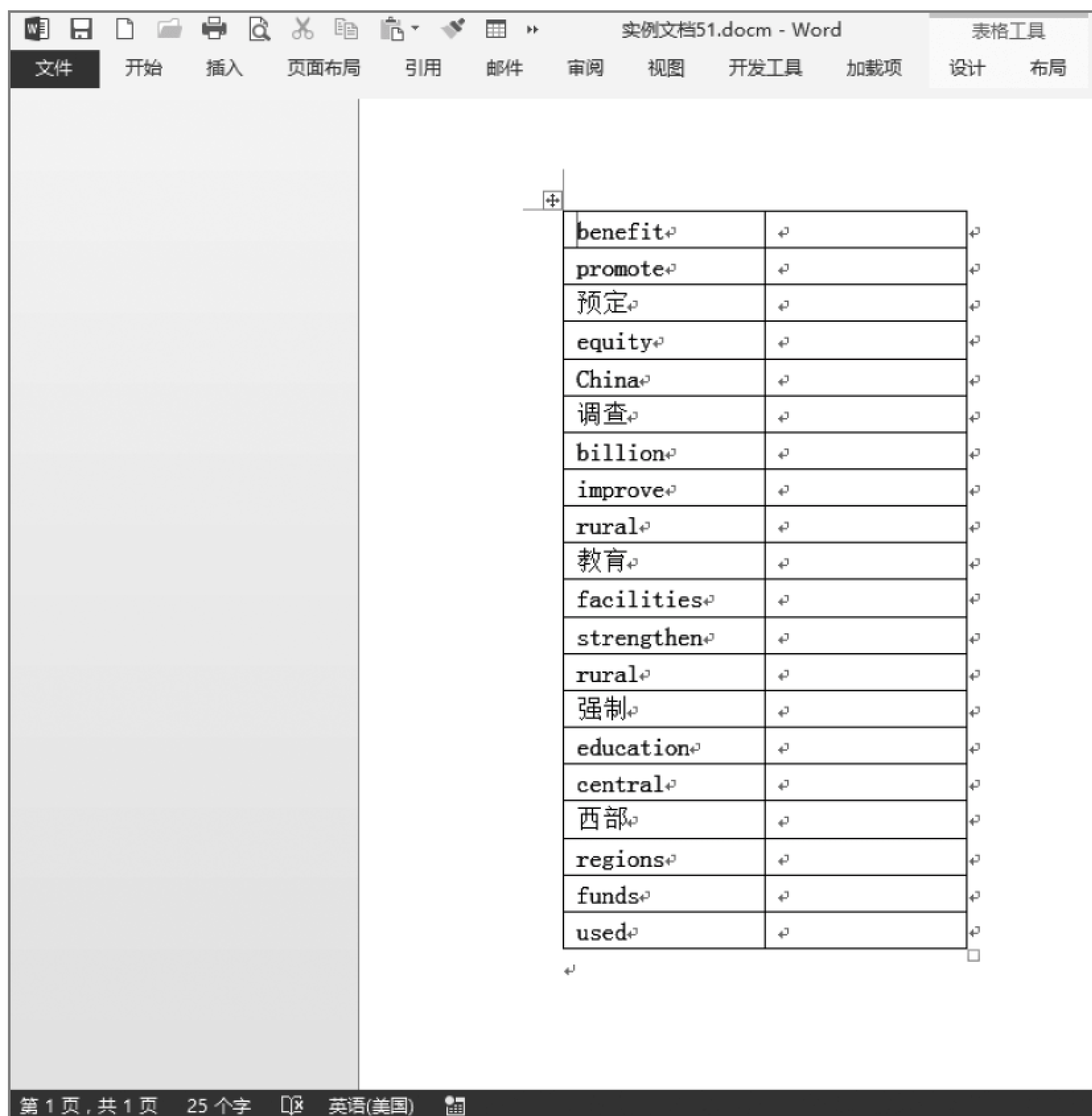


图 9-9 需要翻译的词汇

接下来创建一个 Excel 工作簿“翻译模板 .xlsm”，在其单元格 B1 输入一个公式：

```
=FILTERXML(WEBSERVICE("http://fanyi.youdao.com/translate?&i="&A1&"&doctype=xml
&version"),"//translation")
```

这个公式的作用是翻译单元格 A1 的内容，如图 9-10 所示。

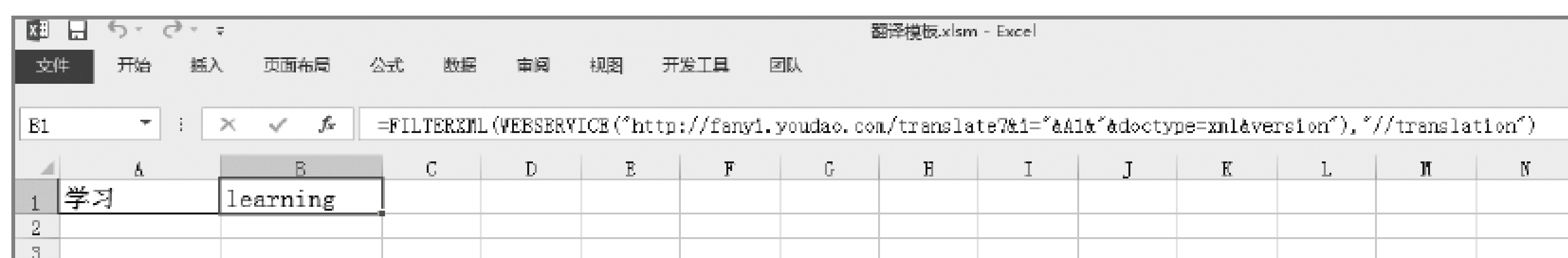


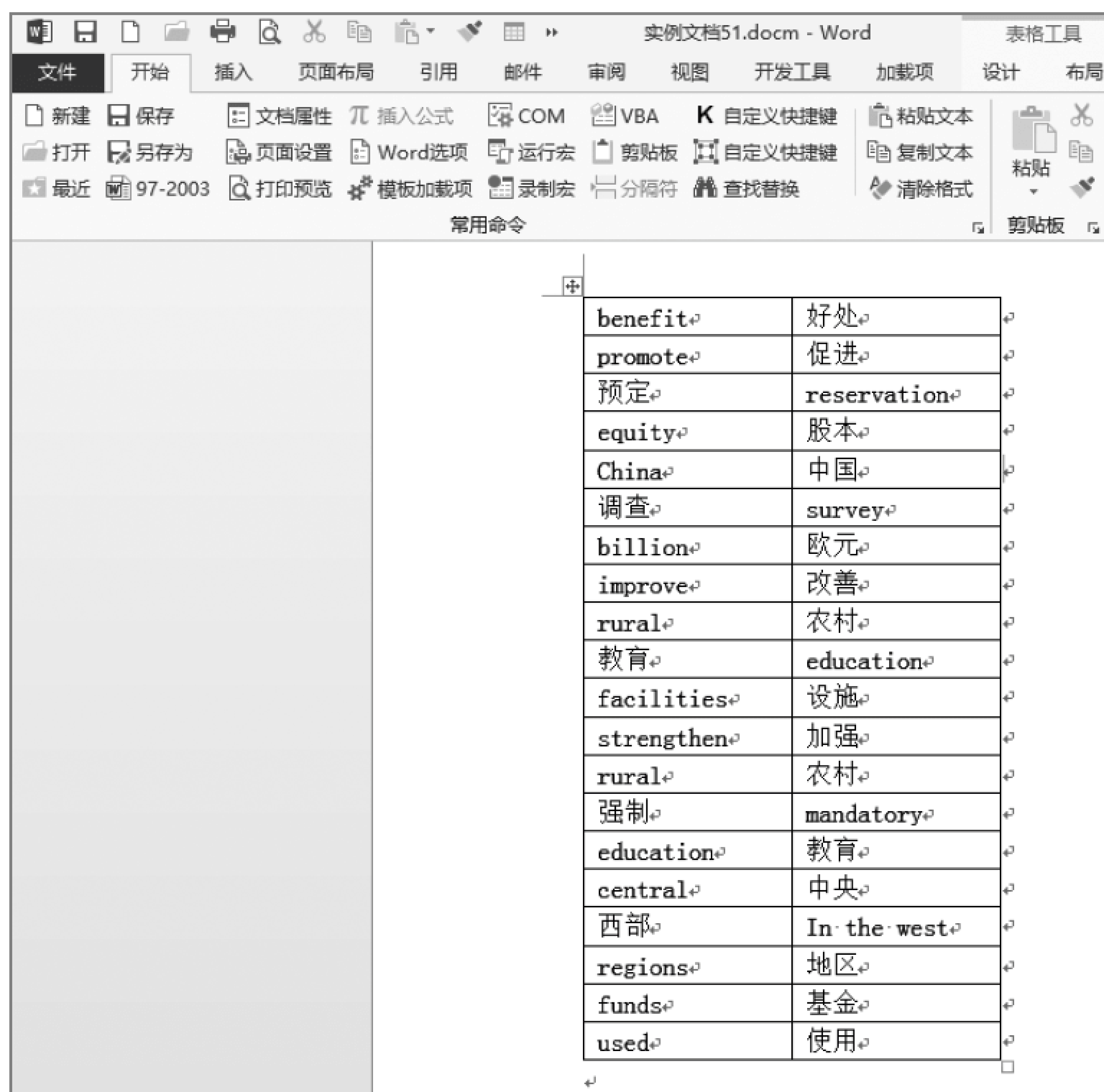
图 9-10 Excel 中的翻译函数

只要修改单元格 A1 的内容，B1 会自动变为其对应的译文。

接下来在 Word VBA 中创建如下过程，实现原理是把 Word 表格中每一个词汇依次发送到 Excel 的 A1 单元格中，待其翻译完毕后，再把 Excel 的 B1 单元格译文发送回 Word 表格第 2 列中。

```
Sub Translate()
    Dim ExcelApp As Excel.Application
    Dim i As Integer
    Dim table As Word.table                        '声明一个 Word 表格对象
    Set ExcelApp = GetObject(, "Excel.Application")    '获取运行中的 Excel
    Set table = Application.ActiveDocument.Tables(1) 'table 表示文档中第 1 个表格
    With ExcelApp
        For i = 1 To table.Rows.Count              '遍历 Word 表格的行，从第 1 行到最末行
            Debug.Print table.Cell(i, 1).Range.Text
            .Range("A1").Value = table.Cell(i, 1).Range.Text
                             '把 Word 表格第 i 行、第 1 列内容发到 Excel 的 A1 单元格
            table.Cell(i, 2).Range.Text = .Range("B1").Value
                             '把 Excel 的 B1 单元格（翻译结果）发送到 Word 表格第 i 行、第 2 列
        Next i
    End With
End Sub
```

运行上述过程，可以看到“实例文档 51.xlsm”表格的第 2 列自动填充完毕，如图 9-11 所示。



benefit	好处
promote	促进
预定	reservation
equity	股本
China	中国
调查	survey
billion	欧元
improve	改善
rural	农村
教育	education
facilities	设施
strengthen	加强
rural	农村
强制	mandatory
education	教育
central	中央
西部	In the west
regions	地区
funds	基金
used	使用

图 9-11　借助 Excel 的翻译功能翻译 Word 表格内容

9.4.2　PowerPoint VBA 调用 Excel VBA 实现自动计算

Excel VBA 中 Application 的 Evaluate 可以对表达式进行求值。例如 Application. Evaluate("30-8/2") 就可以返回 26。

利用这个特点，可以把这个计算功能引入 PowerPoint 中，在幻灯片上放入两个文本框，

一个用来输入表达式，另一个用于存放计算的结果。

单击 PowerPoint 的【开发工具 / 控件】，把一个 CommandButton 放在幻灯片上，并且把该命令按钮的标题改为“计算”，如图 9-12 所示。

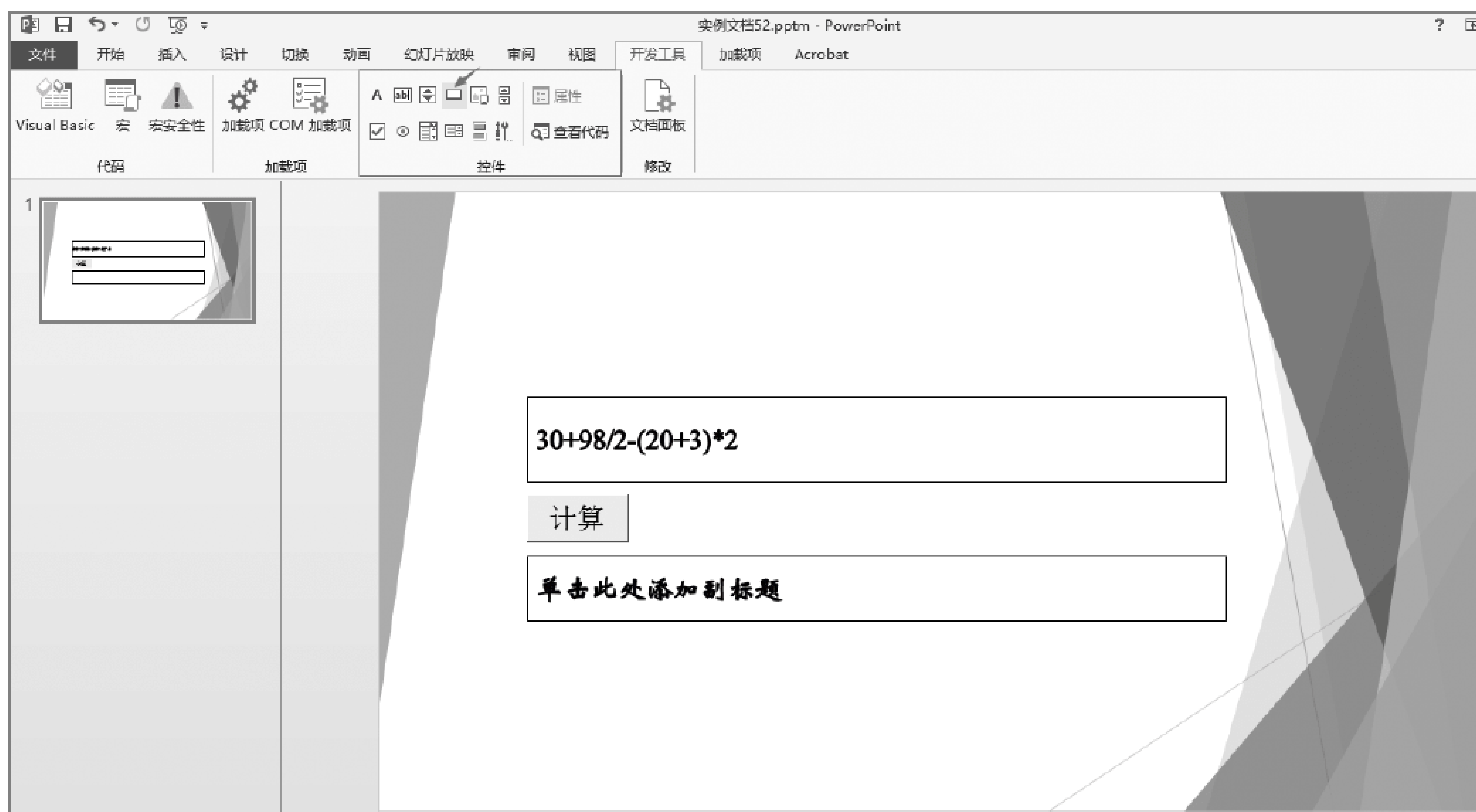


图 9-12　PowerPoint VBA 设计

然后双击该命令按钮，编写其 Click 事件。

```
    Private Sub CommandButton1_Click()
        Dim ExcelApp As Excel.Application
        Dim result As Double
        Set ExcelApp = GetObject(, "Excel.Application")
        result = ExcelApp.Evaluate(Application.ActivePresentation.Slides(1).Shapes(1).
TextFrame.TextRange.Text)
        Application.ActivePresentation.Slides(1).Shapes(2).TextFrame.TextRange.Text =
result
        Set ExcelApp = Nothing
    End Sub
```

代码分析：对于 PowerPoint VBA，文本框也是幻灯片上的 Shape 对象，因此使用 Shapes(1).TextFrame.TextRange.Text 得到第一个文本框的内容，使用 Excel VBA 的 Evaluate 方法对其求值，把 result 返回到第二个文本框中。

写好如上代码后，让幻灯片进入放映模式，点击“计算”按钮，可以看到自动返回计算结果，如图 9-13 所示。

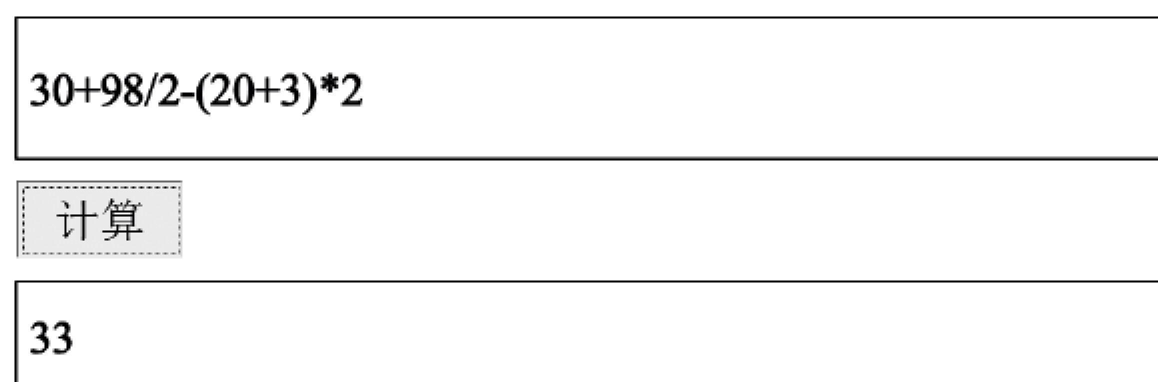


图 9-13　借助 Excel VBA 的 Evaluate 自动计算表达式的值

9.4.3 Outlook VBA 基于 Excel 数据发送邮件

利用 Outlook VBA 可以方便地进行自动收发邮件，如果以 Excel 工作表数据作为数据源，可以实现批量发送邮件。

首先在 Excel 中创建基础数据，假设要给多个面试人员发邮件，如图 9-14 所示。

收件人	主题	正文	附件
lyflyf715@sina.com	面试通知	很抱歉，您没有通过这次面试。	
32669315@qq.com	面试通知	您好！恭喜您通过面试，请于下周一报到！	C:\temp\新员工入职须知.doc

图 9-14 用于发送邮件的基础数据

然后在 Outlook 中打开 VBA 编程界面，插入一个标准模块，并添加对 Excel 对象库的引用。

标准模块中的代码如下。

```
Sub 批量发信()
    Dim ExcelApp As Excel.Application
    Dim Mail As Outlook.MailItem
    Dim i As Integer
    Set ExcelApp = GetObject(, "Excel.Application")
    For i = 2 To 3
        Set Mail = Application.CreateItem(ItemType:=Outlook.OlItemType.olMailItem)
        With Mail
            .To = ExcelApp.Range("A" & i).Value
            .Subject = ExcelApp.Range("B" & i).Value
            .Body = ExcelApp.Range("C" & i).Value
            If IsEmpty(ExcelApp.Range("D" & i)) = False Then
                .Attachments.Add ExcelApp.Range("D" & i).Value
            End If
            '.Display
            .Send
        End With
    Next i
End Sub
```

代码分析：Outlook 的 MailItem 代表一封邮件，属性 To、Subject、Body、Attachment 分别代表邮件的收件人、主题、正文、附件。

运行上述过程，从 Outlook 的已发送邮件可以看到发出去的邮件，当然面试人员的收件箱也能收到这些邮件，如图 9-15 所示。

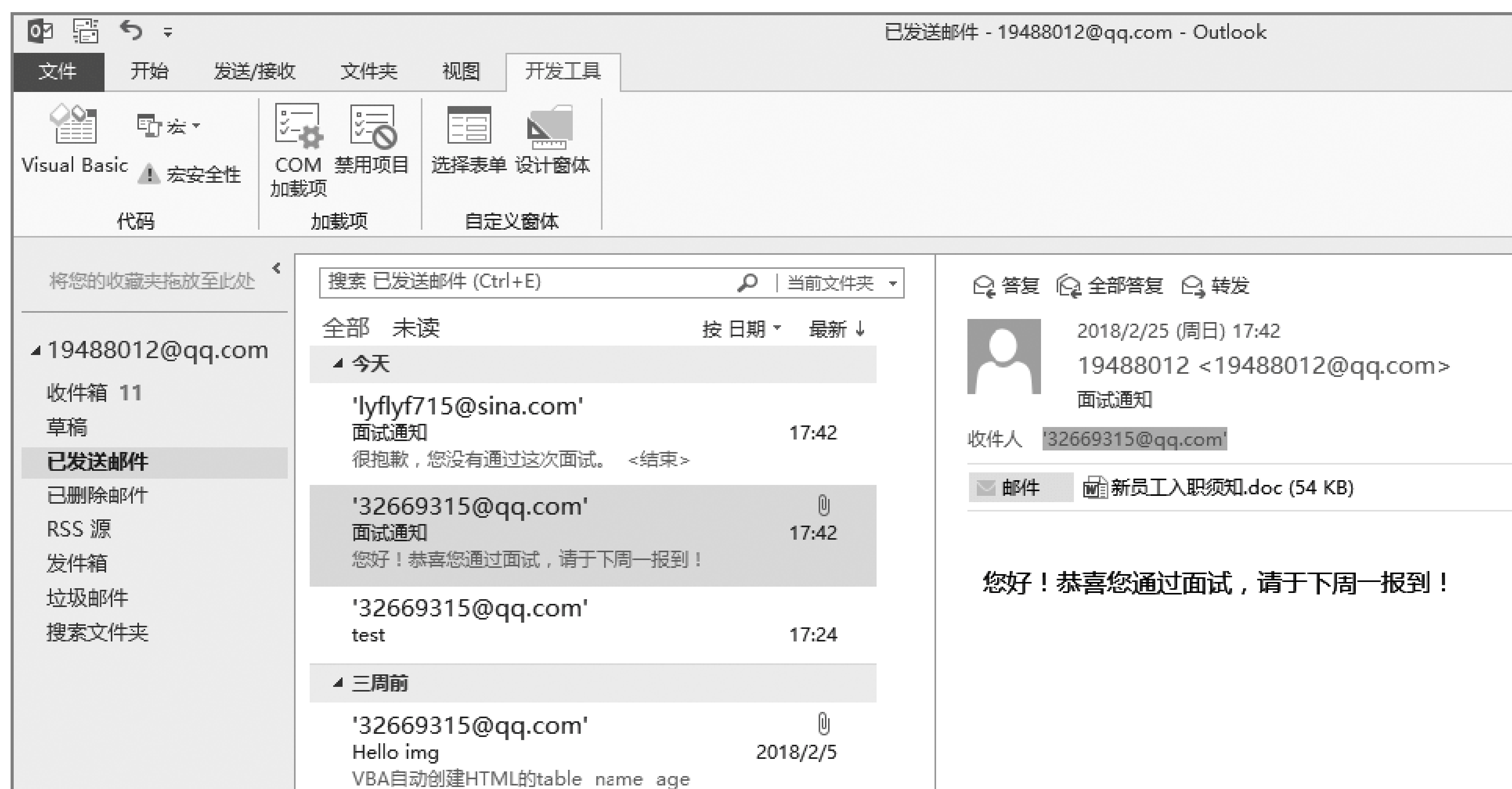


图 9-15 批量发送邮件

9.4.4 Visual Basic 6.0 读写 Excel

Visual Basic 6.0 也可以方便地操作访问微软 Office，下面的实例把 VB 窗体中的数据发到 Excel 单元格中。

启动 Visual Basic 6.0，创建一个应用程序项目，窗体上放置一个 Text 控件，用于输入起始路径，再放一个 File 控件，用来自动显示该路径下的所有文件名称，再放一个 Command 命令按钮，如图 9-16 所示。

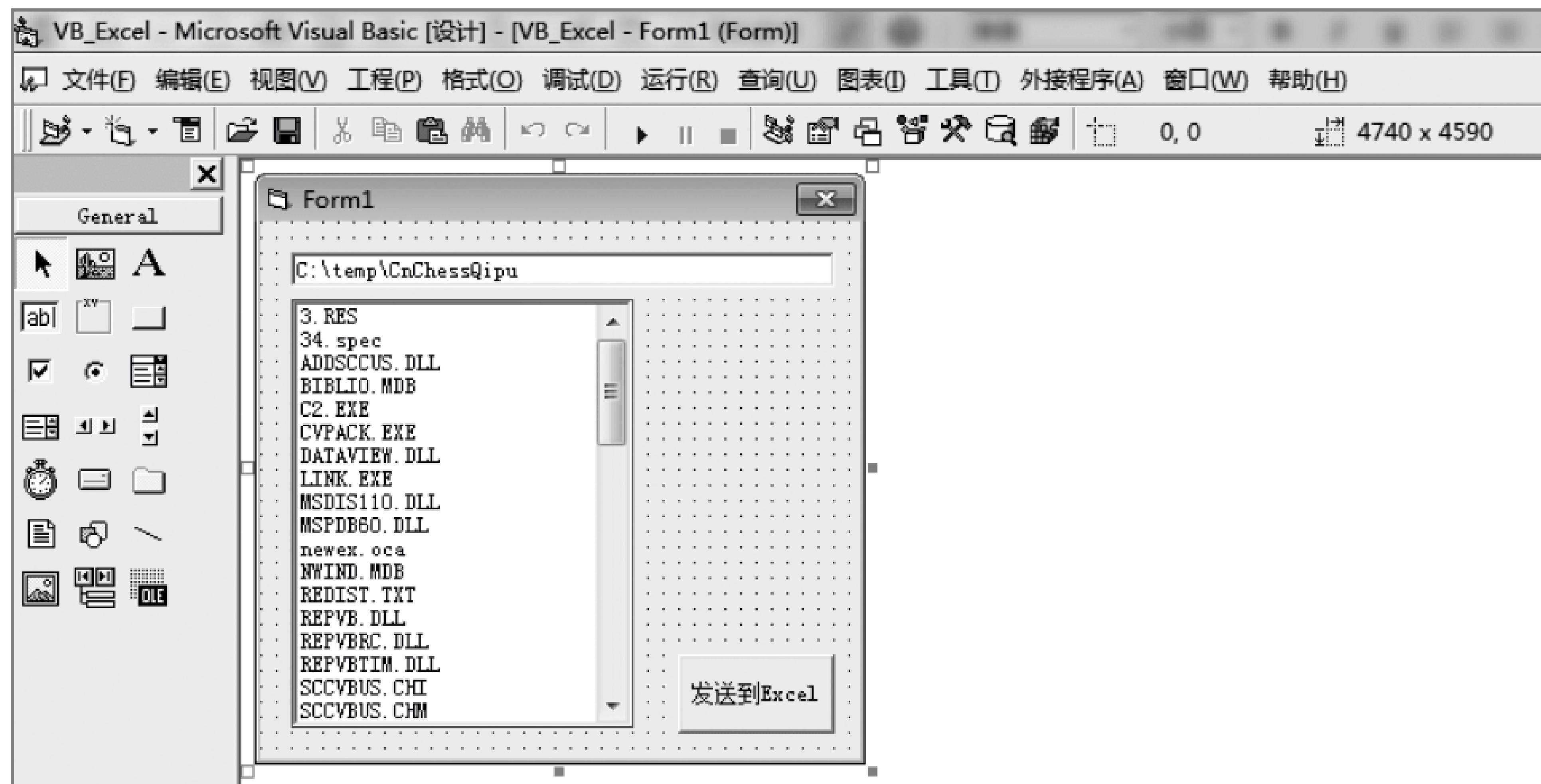


图 9-16 VB6 中的窗体设计视图

程序的意图是，把 File 控件列出的文件名称发送到 Excel 单元格中。

因此，单击 VB 6.0 的菜单【工程 / 引用】，添加对 Excel 的引用。

然后双击窗体中的“发送到 Excel”按钮，进入 Form1 的代码窗口，输入 Form 的 Load 事件，以及按钮的 Click 事件。

```
Private Sub Command1_Click()
    Dim ExcelApp As Excel.Application
    Dim i As Integer
    Set ExcelApp = GetObject(, "Excel.Application")
    ExcelApp.Workbooks.Add
    For i = 0 To Me.File1.ListCount - 1
        ExcelApp.Wait Now + TimeValue("00:00:01")
        Me.File1.ListIndex = i
        ExcelApp.Range("A" & i + 1).Value = Me.File1.List(i) 'Excel 的行号从 1 开始，所以 +1
    Next i
End Sub

Private Sub Form_Load()
    Me.File1.Path = Me.Text1.Text
End Sub
```

启动窗体后，单击“发送到 Excel”按钮，会看到隔一秒就发送一条数据到 Excel 的 A 列中，如图 9-17 所示。

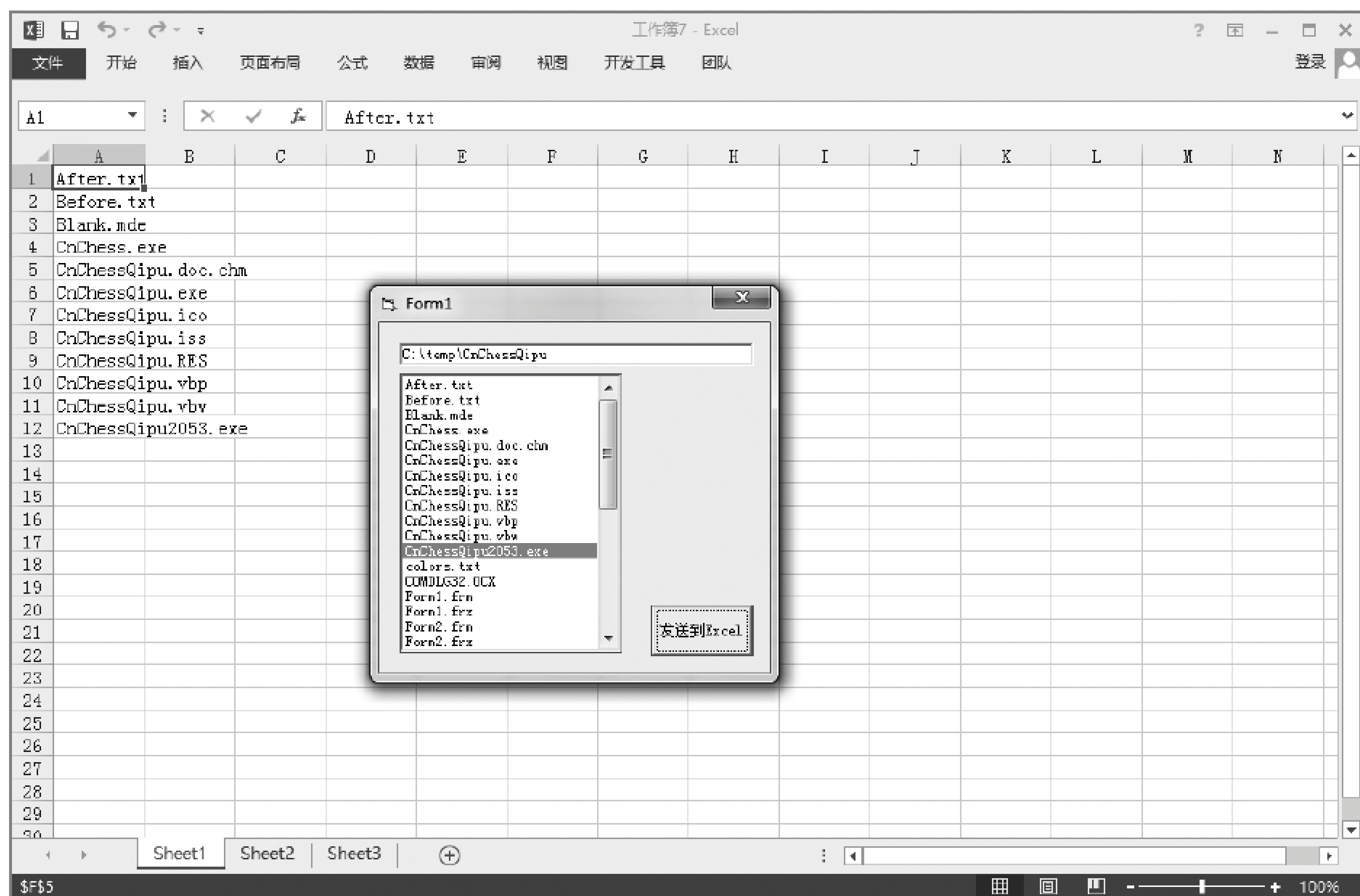


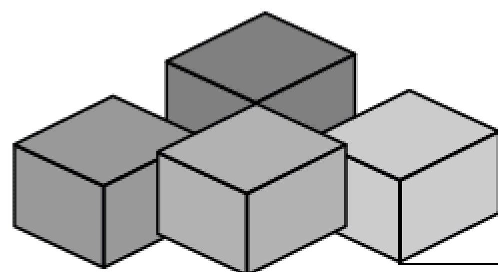
图 9-17　VB 程序中的数据自动发送到 Excel 单元格中

以上案例的工程源代码为“VB 操作 Excel/VB_Excel.vbp”。

9.5　本章小结

前期绑定不是必需的，即使在 VBA 工程中不添加引用，也能实现程序同样的功能和结果。然而，不添加引用的情况下，书写代码非常不方便，所有对象均需声明为 Object，而不是具体的对象类型，而且不能使用外部引用中的常量。

GetObject、New、CreateObject 用于获取和创建外部对象，从而达到调用其他 Office 组件的目的。



第 10 章 工程引用与外部对象

本章介绍使用 VBA 遍历、读写 VBA 工程中引用的方法。

本章用到的外部引用和重要对象：

- ❑ lMicrosoft Visual Basic for Applications Extensibility 5.3
  - ➢ VBIDE.Reference
  - ➢ VBIDE.VBProject

10.1 处理 VBA 工程中的引用

VBA 程序是以工程（VBProject）为单位的，对于 Excel VBA，一个工作簿有且只有一个独立的 VBA 工程，一个工程下面由普通模块、类模块、窗体等构成，可以通过 VBA 编程环境的工程资源管理器窗格看到，如图 10-1 所示。



图 10-1 VBA 工程资源管理器

每个 VBA 工程有一个引用集合（References），可以单击 VBA 编程环境的菜单【工具 / 引用】，弹出引用对话框，引用对话框中最上面的并且处于勾选状态的就是当前工程的引用集合。

实际上，VBA 允许通过程序代码的方式来访问 VBE 中的各个 VBA 工程，以及 VBA 工程中的各个引用。例如，运行如下两行代码。

```
Msgbox Application.VBE.VBProjects.Count
Msgbox Application.VBE.ActiveVBProject.References.Count
```

结果弹出“运行时错误 1004”错误对话框，如图 10-2 所示。

这是因为 Office 的宏安全性对话框中默认不勾选“信任对 VBA 工程对象模型的访问”，如图 10-3 所示。

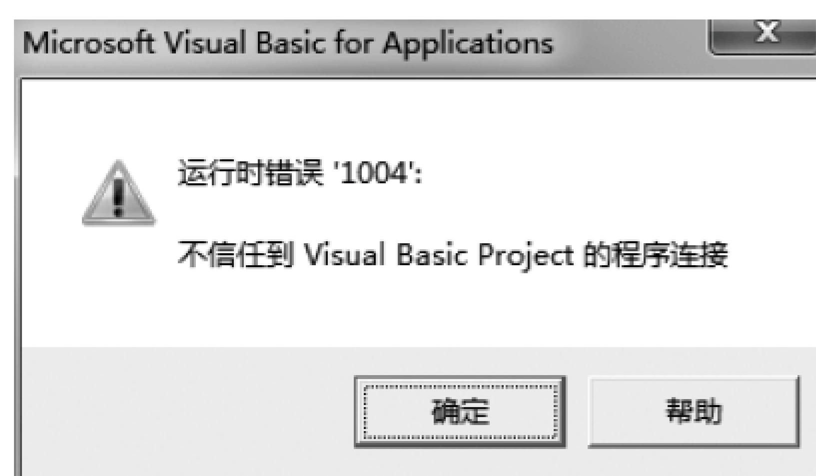


图 10-2　不允许访问 VBA 工程对象模型

因此，凡是涉及使用 VBA 操作 VBA 工程方面的、VBIDE 对象方面的，必须事先勾选上述安全性选项。

为了便于使用 VBIDE 中的对象类型和常数，在操作 VBA 工程的程序中需要添加“Microsoft Visual Basic for Applications Extensibility 5.3”，如图 10-4 所示。

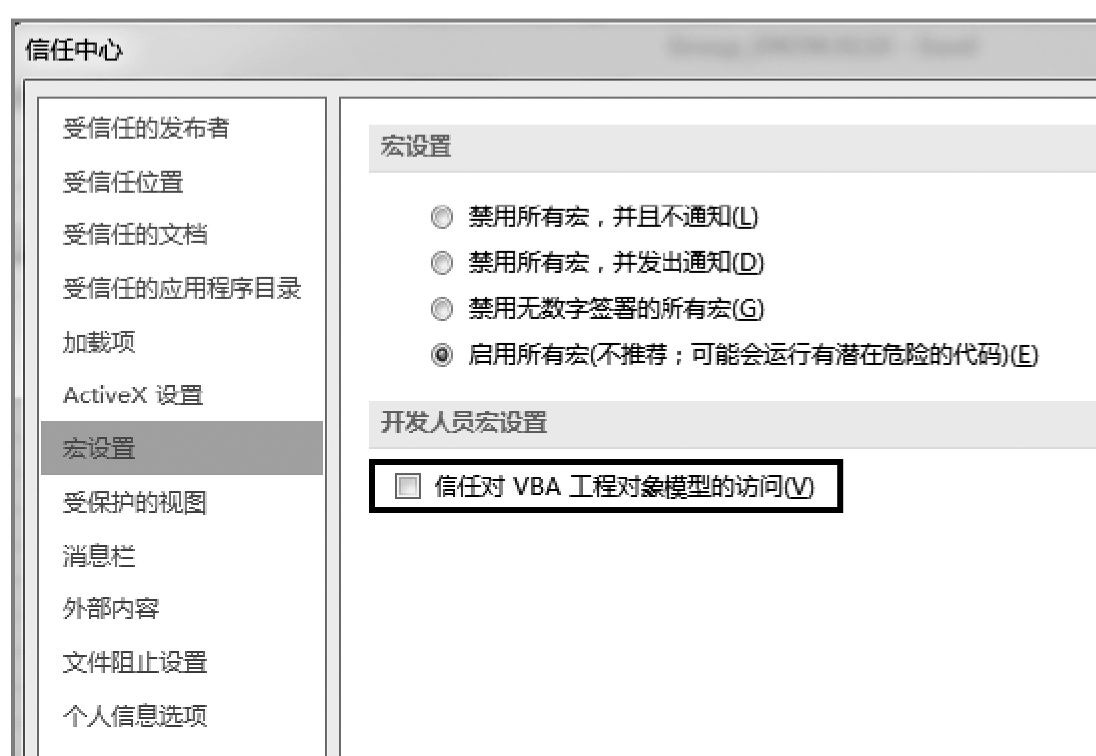


图 10-3　安全性选项对话框

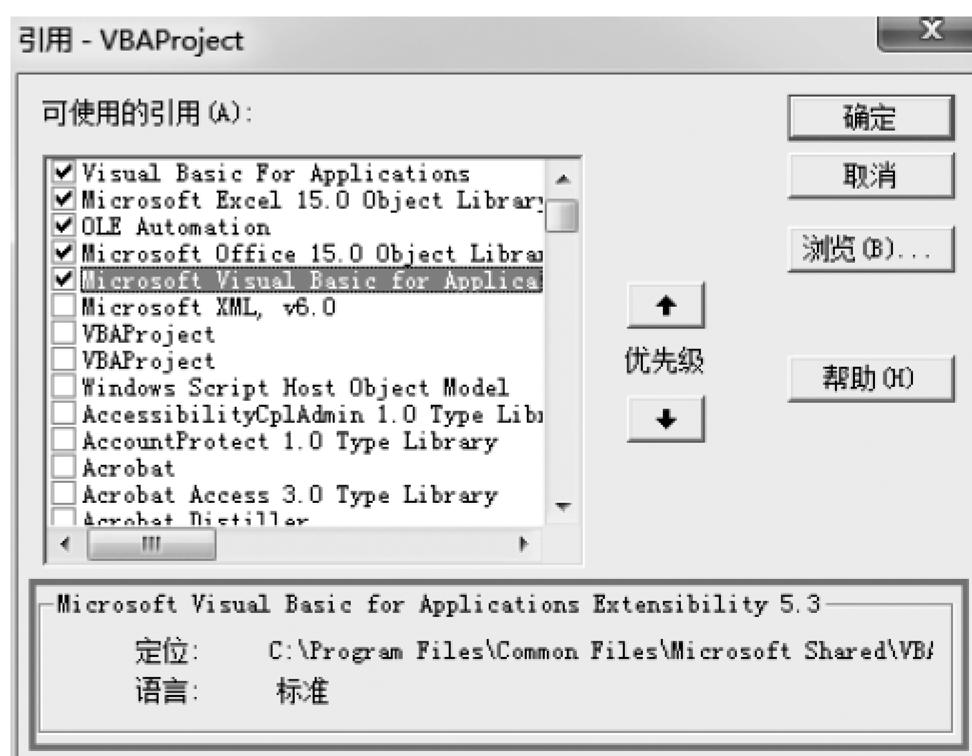


图 10-4　添加外部引用

添加该引用后，在代码中可以声明以 VBIDE 开头的对象类型。

10.1.1　引用的属性

VBA 工程中的一个引用其实是建立了 VBA 工程与外部动态链接库文件的一种链接关系，假设添加“Microsoft Scripting Runtime”这个引用，该引用的名称是“Scripting”，描述是“Microsoft Scripting Runtime”，完全路径是“C:\Windows\System32\scrrun.dll”，如图 10-5 所示。

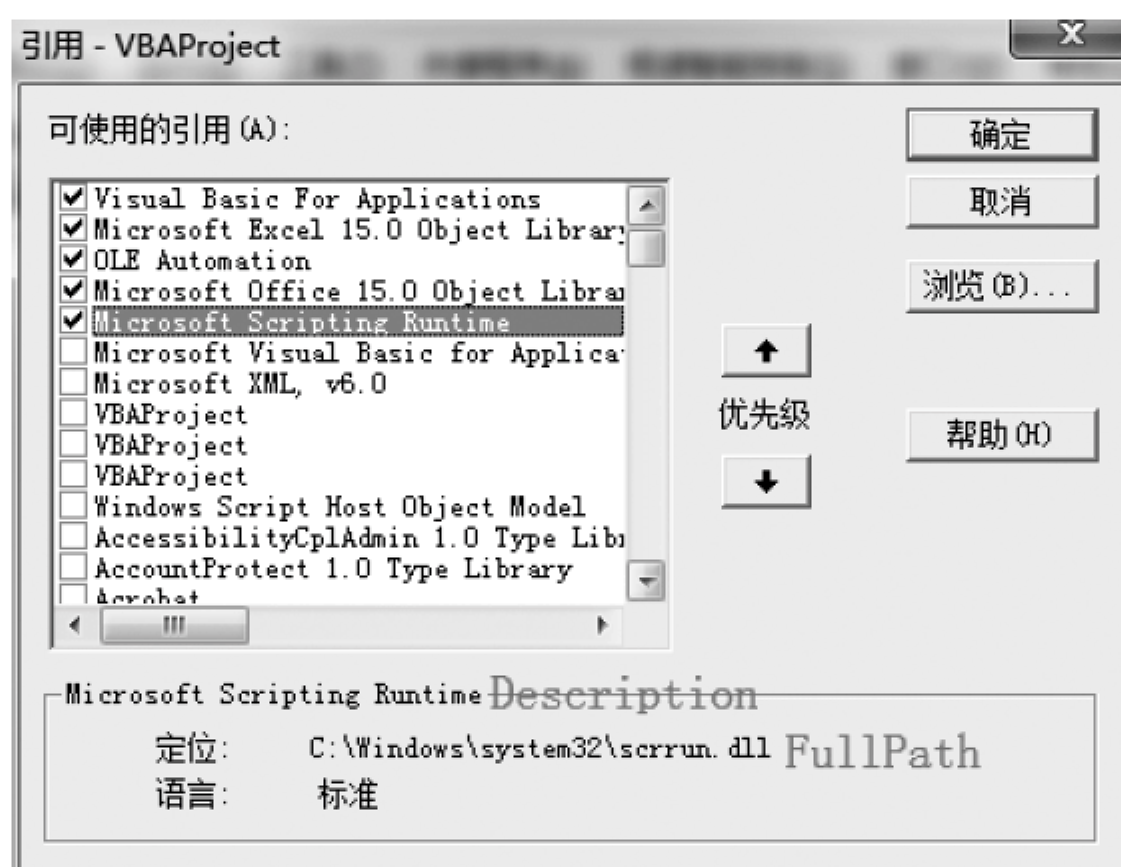


图 10-5　引用的描述和完全路径

另外，引用的对象库在系统的注册表中均有记录，具体位置如下。

```
HKEY_CLASSES_ROOT\TypeLib\
```

例如，展开{420B2830-E718-11CF-893D-00A0C9054228}下面的1.0注册表项，右侧窗格可以看到描述文本为“Microsoft Scripting Runtime”。其中，{420B2830-E718-11CF-893D-00A0C9054228}就是这个引用库文件的GUID。1.0表示这个动态链接库文件的版本号，主版本号为1，次版本号为0，如图10-6所示。

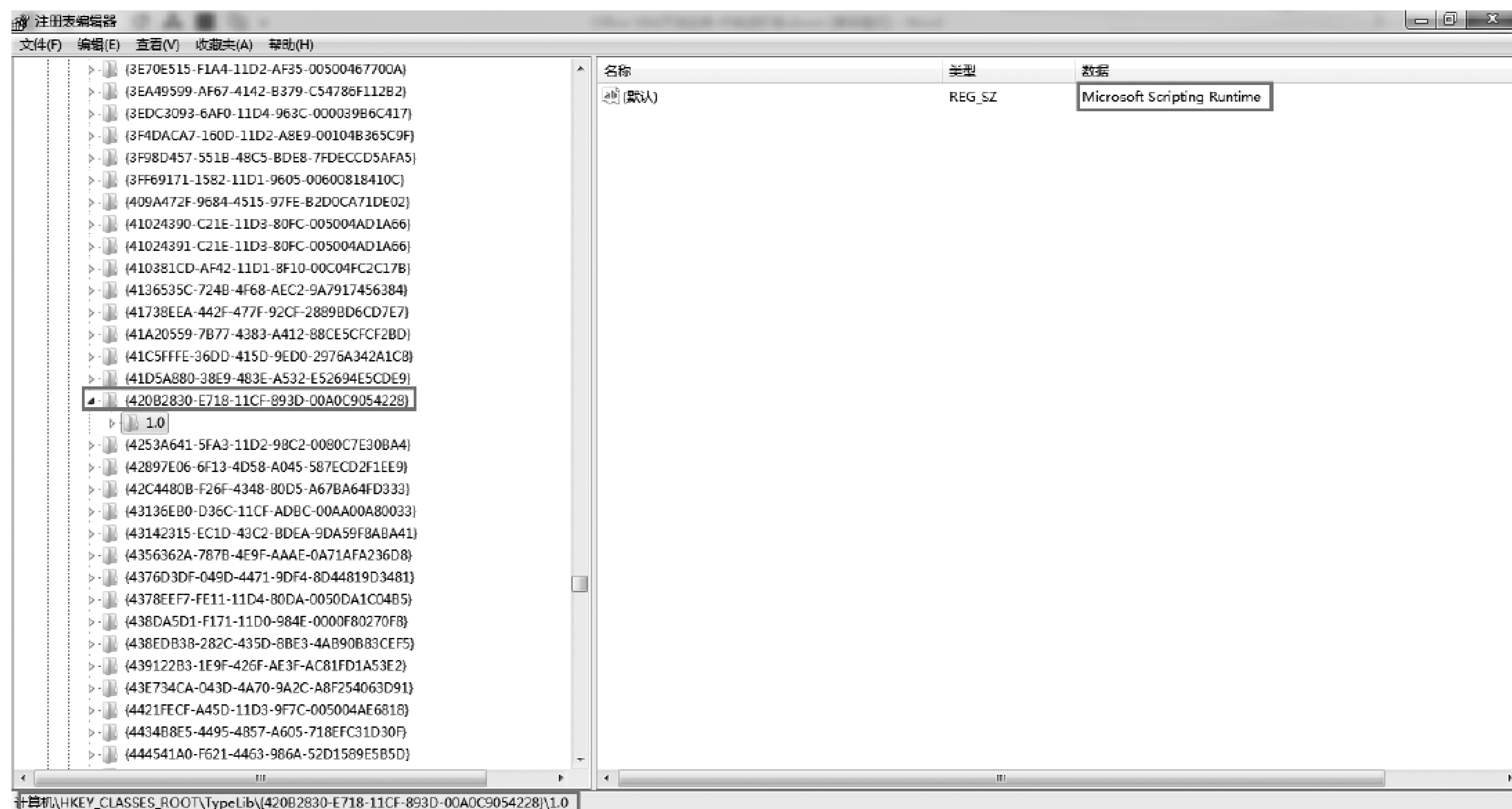


图 10-6 引用对象库的 GUID

因此，VBA工程中引用的常用属性如下。

❑ Name：引用的名称。
❑ Description：引用的描述。
❑ GUID：注册表中的存储信息。
❑ Major：主版本号。
❑ Minor：次版本号。
❑ FullPath：动态链接库的路径。

运行如下程序，可以打印VBA工程中每个引用的属性。

```
Sub 遍历所有引用 ()
    Dim ref As VBIDE.Reference
    ' Application.ActiveWorkbook.VBProject.References '活动工作簿的所有引用
    For Each ref In Application.VBE.ActiveVBProject.References
        Debug.Print ref.Name, ref.Description, ref.GUID, ref.Major, ref.Minor,
ref.FullPath
    Next ref
End Sub
```

代码分析：Application.ActiveWorkbook.VBProject表示Excel的活动工作簿的VBA工

程，而 Application.VBE.ActiveVBProject 表示 VBE 中的活动工程，不是同一个概念。

VBA 编程过程中，使用最频繁的引用如表 10-1 所示。

表 10-1　VBA 最常用的外部引用

名　　称	描　　述	GUID	主版本号	次版本号
Excel	Microsoft Excel 15.0 Object Library	{00020813-0000-0000-C000-000000000046}	1	8
Word	Microsoft Word 15.0 Object Library	{00020905-0000-0000-C000-000000000046}	8	6
PowerPoint	Microsoft PowerPoint 11.0 Object Library	{91493440-5A91-11CF-8700-00AA0060263B}	2	8
Outlook	Microsoft Outlook 15.0 Object Library	{00062FFF-0000-0000-C000-000000000046}	9	5
Office	Microsoft Office 15.0 Object Library	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}	2	7
VBIDE	Microsoft Visual Basic for Applications Extensibility 5.3	{0002E157-0000-0000-C000-000000000046}	5	3
MSForms	Microsoft Forms 2.0 Object Library	{0D452EE1-E08F-101A-852E-02608C4D0BB4}	2	0
VBScript_RegExp_55	Microsoft VBScript Regular Expressions 5.5	{3F4DACA7-160D-11D2-A8E9-00104B365C9F}	5	5
Scripting	Microsoft Scripting Runtime	{420B2830-E718-11CF-893D-00A0C9054228}	1	0
MSXML2	Microsoft XML, v6.0	{F5078F18-C551-11D3-89B9-0000F81FE221}	6	0
IWshRuntimeLibrary	Windows Script Host Object Model	{F935DC20-1CF0-11D0-ADB9-00C04FD58A0B}	1	0
WinHttp	Microsoft WinHTTP Services, version 5.1	{662901FC-6951-4854-9EB2-D9A2570F2B2E}	5	1
MSHTML	Microsoft HTML Object Library	{3050F1C5-98B5-11CF-BB82-00AA00BDCE0B}	4	0
SHDocVw	Microsoft Internet Controls	{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}	1	1
ADODB	Microsoft ActiveX Data Objects 2.8 Library	{2A75196C-D9EB-4129-B803-931327F72D5C}	2	8
ADOX	Microsoft ADO Ext. 2.8 for DDL and Security	{00000600-0000-0010-8000-00AA006D2EA4}	2	8
MSScriptControl	Microsoft Script Control 1.0	{0E59F1D2-1FBE-11D0-8FF2-00A0D10038BC}	1	0
Shell32	Microsoft Shell Controls And Automation	{50A7E9B0-70EF-11D1-B75A-00A0C90564FE}	1	0
WbemScripting	Microsoft WMI Scripting V1.2 Library	{565783C6-CB41-11D1-8B02-00600806D9B6}	1	2
CDO	Microsoft CDO for Windows 2000 Library	{CD000000-8B95-11D1-82DB-00C04FB1625D}	1	0
Acrobat	Adobe Acrobat 9.0 Type Library	{E64169B3-3592-47D2-816E-602C5C13F328}	1	1

10.1.2 内置引用

任何一个 VBA 工程都有一些内置引用，例如 Excel VBA 中的内置引用是 Visual Basic for Applications 和 Microsoft Excel x.0 Object Library。这些内置引用的 Builtin 属性为 True，而且不能移除内置引用。

下面的程序可以批量移除工作簿的 VBA 工程中的非内置引用。

```
Sub 移除所有非内置引用 ()
    Dim ref As VBIDE.Reference
    For Each ref In Application.ActiveWorkbook.
VBProject.References
        If ref.BuiltIn = False Then
            Application.ActiveWorkbook.
VBProject.References.Remove ref
        End If
    Next ref
End Sub
```

运行上述程序，引用对话框中只有两个内置引用仍处于勾选状态，如图 10-7 所示。

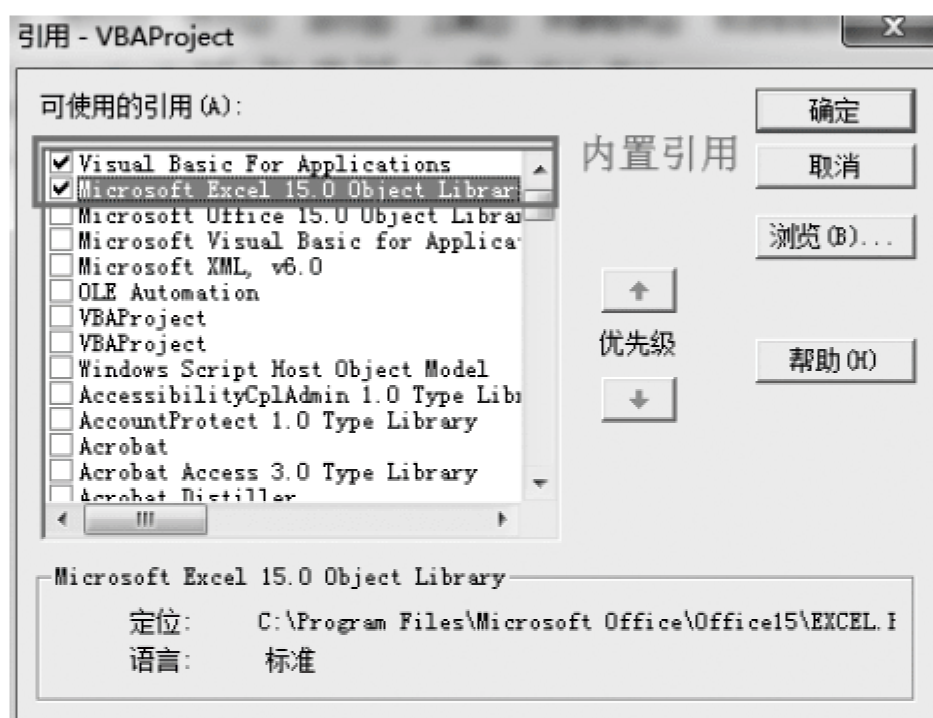


图 10-7 Excel VBA 工程中的两个内置引用

10.1.3 引用的添加

使用代码自动为 VBA 工程添加引用，可以使用 AddFromFile、AddFromGuid 两种方法。AddFromFile 方法需要提供外部引用文件的路径，AddFromGuid 方法需要提供外部引用库的 GUID 值和版本号。

例如，下面的程序分别使用以上两种方法，自动为活动工作簿的 VBA 工程添加 XML v6.0、PowerPoint 11.0 的引用。

```
Sub 引用的添加 ()
    Dim ref As VBIDE.Reference
     Set ref = Application.ActiveWorkbook.VBProject.References.AddFromFile
(Filename:="C:\Windows\System32\msxml6.dll") '添加 XML v6.0
    Set ref = Application.ActiveWorkbook.VBProject.References.AddFromGuid (GUID:
="{91493440-5A91-11CF-8700-00AA0060263B}", Major:=2, Minor:=8) '添加 PowerPoint 11.0
End Sub
```

注意 同一个引用不能重复添加多次，也不能添加不同版本的同一引用。例如，不能在同一个工程同时添加 Office 11.0 和 Office 15.0。

10.1.4 引用的移除

使用 Remove 方法可以移除工程中已存在的引用，但是移除之前必须先知道该引用的名称（是 Name，不是 Description），例如 Microsoft XML V6.0 这个引用的名称是 MSXML2，OLE Automation 这个引用的名称是 stdole。

下面的程序移除活动工作簿的 VBA 工程中的两个引用。

```
Sub 引用的移除 ()
    Dim ref As VBIDE.Reference
    Set ref = Application.ActiveWorkbook.VBProject.References.Item("MSXML2")
    Application.ActiveWorkbook.VBProject.References.Remove Reference:=ref
                                            ' 移除 Microsoft XML v6.0
    Set ref = Application.ActiveWorkbook.VBProject.References.Item("stdole")
    Application.ActiveWorkbook.VBProject.References.Remove Reference:=ref
                                            ' 移除 OLE Automation
End Sub
```

注意　内置引用不能被移除，例如 Word VBA 工程中不能移除 Microsoft Word Object Library。

以上程序的源代码文件为“实例文档 53.xlsm”。

10.2　外部对象和注册表

Windows 系统的注册表中，不仅存储着各个外部引用库的 GUID，而且引用库中的对象和类也都有相应的 CLSID。

在注册表编辑器中，展开如下注册表项。

```
HKEY_CLASSES_ROOT\CLSID
```

可以看到下面包含了很多用花括号括起来的子项。

例如展开 {3F4DACA4-160D-11D2-A8E9-00104B365C9F} 这个子项，可以看到 ProgID 的值为 VBScript.RegExp，如图 10-8 所示。

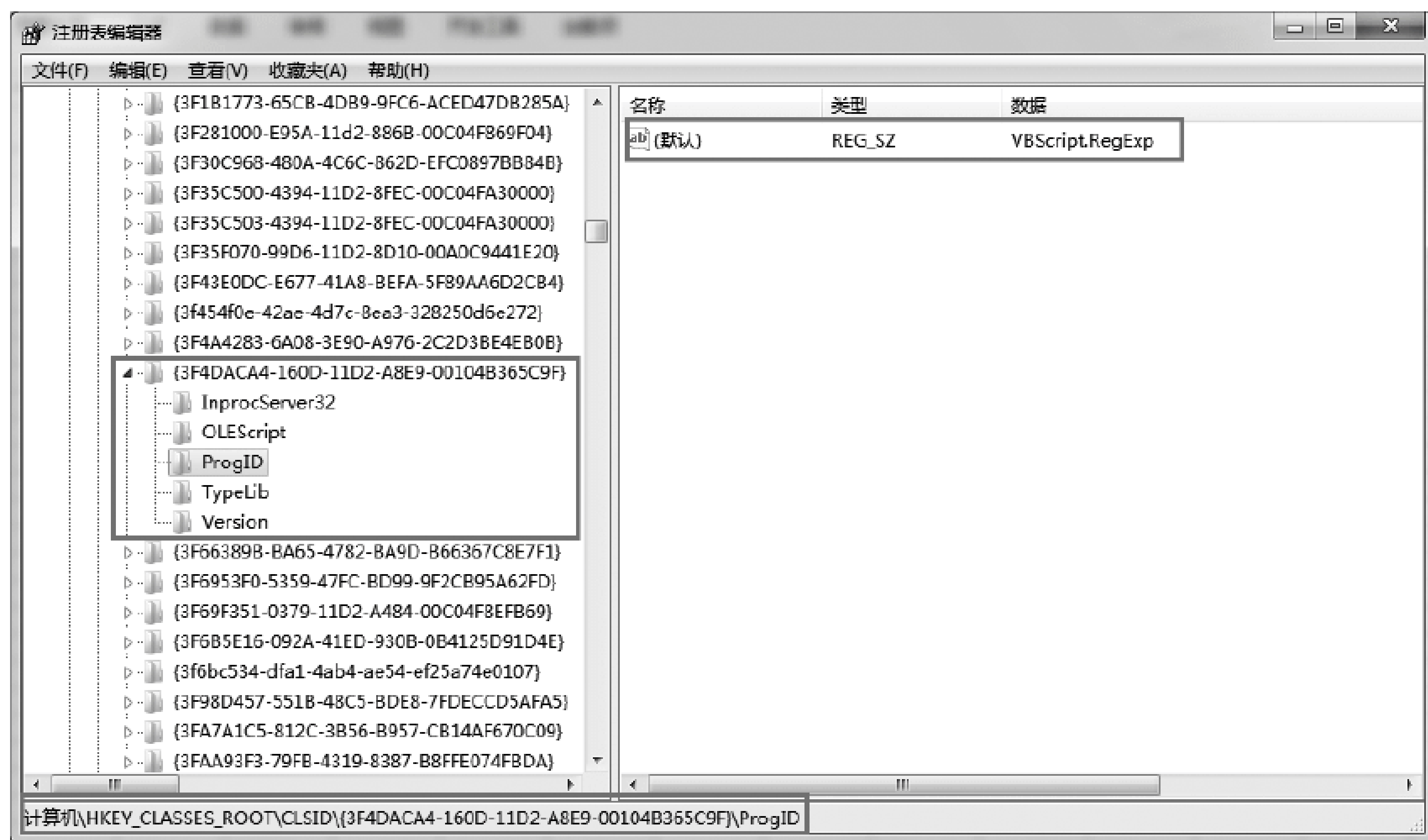


图 10-8　正则表达式对象库的 CLSID 和 ProgID

因此可以看出，正则表达式对象的 ProgID 是 VBScript.RegExp，CLSID 是 {3F4DACA4-160D-11D2-A8E9-00104B365C9F}。

其他常见对象的 CLSID、ProgID 也可以从注册表中找到。

10.2.1 CLSID 和 ProgID

CLSID 是指 Windows 系统对于不同的应用程序、文件类型、OLE 对象、特殊文件夹以及各种系统组件分配的一个唯一表示它的 ID 代码，用于对其身份的标识和与其他对象进行区分。

ProgID 是程序员给某个 CLSID 指定的一个易记的名字。

VBA 编程常用对象的 CLSID、ProgID 如表 10-2 所示。

表 10-2 VBA 常用对象的 CLSID、ProgID

对　　象	引　　用	前期绑定	ProgID	CLSID
Excel 应用程序	Microsoft Excel 15.0 Object Library	Excel.Application	Excel.Application.15	{00024500-0000-0000-C000-000000000046}
Word 应用程序	Microsoft Word 15.0 Object Library	Word.Application	Word.Application.15	{000209FF-0000-0000-C000-000000000046}
Powerpoint 应用程序	Microsoft PowerPoint 15.0 Object Library	Powerpoint.Application	Powerpoint. Application.15	{91493441-5A91-11CF-8700-00AA0060263B}
Outlook 应用程序	Microsoft Outlook 15.0 Object Library	Outlook.Application	Outlook. Application.15	{0006F03A-0000-0000-C000-000000000046}
正则表达式	Microsoft VBScript Regular Expressions 5.5	VBScript_RegExp_55. RegExp	VBScript.RegExp	{3F4DACA4-160D-11D2-A8E9-00104B365C9F}
字典	Microsoft Scripting Runtime	Scripting.Dictionary	Scripting.Dictionary	{EE09B103-97E0-11CF-978F-00A02463E06F}
FSO	Microsoft Scripting Runtime	Scripting. FileSystemObject	Scripting. FileSystemObject	{0D43FE01-F093-11CF-8940-00A0C9054228}
XML 文档	Microsoft XML, v6.0	MSXML2. DOMDocument60	Msxml2. DOMDocument.6.0	{88d96a05-f192-11d4-a65f-0040963251e5}
XMLHTTP	Microsoft XML, v6.0	MSXML2. XMLHTTP60	Msxml2. XMLHTTP.6.0	{88d96a0a-f192-11d4-a65f-0040963251e5}
WshShell	Windows Script Host Object Model	IWshRuntimeLibrary. WshShell	WScript.Shell	{72C24DD5-D70A-438B-8A42-98424B88AFB8}
WinHttp	Microsoft WinHTTP Services, version 5.1	WinHttp. WinHttpRequest	WinHttp. WinHttpRequest.5.1	{2087c2f4-2cef-4953-a8ab-66779b670495}
HTML文档	Microsoft HTML Object Library	MSHTML. HTMLDocument	htmlfile	{25336920-03F9-11cf-8FD0-00AA00686F13}
IE	Microsoft Internet Controls	SHDocVw. InternetExplorer	InternetExplorer. Application	{0002DF01-0000-0000-C000-000000000046}
ADODB. Connection	Microsoft ActiveX Data Objects 2.8 Library	ADODB.Connection	ADODB.Connection	{00000514-0000-0010-8000-00AA006D2EA4}
ADODB. Recordset	Microsoft ActiveX Data Objects 2.8 Library	ADODB.Recordset	ADODB.Recordset	{00000535-0000-0010-8000-00AA006D2EA4}
MSScript Control	Microsoft Script Control 1.0	MSScriptControl. ScriptControl	MSScriptControl. ScriptControl	{0E59F1D5-1FBE-11D0-8FF2-00A0D10038BC}
Shell32	Microsoft Shell Controls And Automation	Shell32.Shell	Shell.Application	{13709620-C279-11CE-A49E-444553540000}

续表

对　　象	引　　用	前期绑定	ProgID	CLSID
Wbem Scripting	Microsoft WMI Scripting V1.2 Library	WbemScripting.SWbemLocator	WbemScripting.SWbemLocator	{76A64158-CB41-11D1-8B02-00600806D9B6}
CDO 配置	Microsoft CDO for Windows 2000 Library	CDO.Configuration	CDO.Configuration	{CD000002-8B95-11D1-82DB-00C04FB1625D}
CDO 邮件	Microsoft CDO for Windows 2000 Library	CDO.Message	CDO.Message	{CD000001-8B95-11D1-82DB-00C04FB1625D}

10.2.2　创建新对象

前期绑定（事先添加引用）的方式，使用 New 关键字创建新对象。

后期绑定的方式，只能使用 CreateObject 创建新对象。使用 CreateObject 创建新对象时，既可以用 ProgID，也可以用 CLSID 作为参数。

例如，下面的程序分别使用前期绑定、ProgID 和 CLSID 创建新的 XMLHTTP 对象。

```
Sub 创建新对象的方式()
    Dim A As Object, B As Object, C As Object
    Set A = New MSXML2.XMLHTTP60                           '前期绑定
    Set B = CreateObject("Msxml2.XMLHTTP.6.0")             '根据 ProgID
    Set C = CreateObject("new:{88d96a0a-f192-11d4-a65f-0040963251e5}") '根据 CLSID
    Debug.Print TypeName(A), TypeOf A Is MSXML2.XMLHTTP60
    Debug.Print TypeName(B), TypeOf B Is MSXML2.XMLHTTP60
    Debug.Print TypeName(C), TypeOf C Is MSXML2.XMLHTTP60
End Sub
```

代码分析：前期绑定的类型名、后期绑定的 ProgID、CLSID 的取值来自于表 10-2。

TypeName 返回类型名称，是一个字符串。TypeOf 用于把一个变量与类型名称做比较，如果类型匹配则返回 True。

运行上述程序，立即窗口分别打印对象变量 A、B、C 的类型信息，如图 10-9 所示。

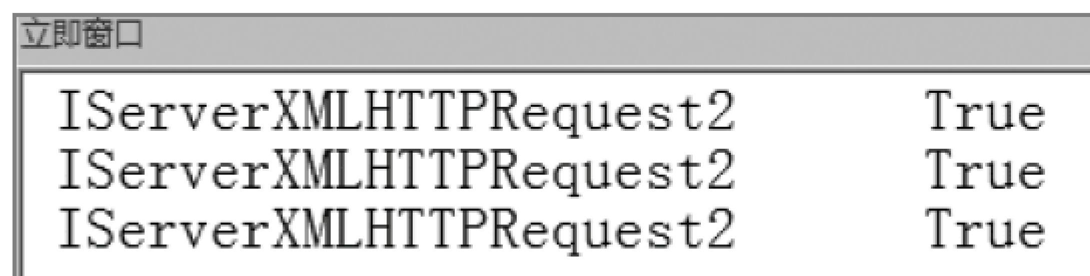


图 10-9　返回相同的对象类型

可以看出，3 个变量的类型完全一样，都是 XMLHTTP 对象。

10.2.3　VBA 中使用剪贴板

在实际编程过程中，经常需要剪贴板的自动化操作，例如把字符串放到剪贴板上，或者从剪贴板上获取数据。

下面的程序使用后期绑定方式，把一个英文句子放入剪贴板。

```
Sub 放入剪贴板 ()
    Dim D As Object
    Set D = CreateObject("new:{1C3B4210-F441-11CE-B9EA-00AA006B1A69}")
    With D
        .Clear
        .SetText "Have a good dream"
        .PutInClipboard
    End With
End Sub
```

运行上述程序后，在任意可以输入文字的地方按下【Ctrl+V】快捷键就可以粘贴如上内容。

反过来，如果从现在的剪贴板上获取文字内容，可以运行如下程序。

```
Sub 获取剪贴板 ()
    Dim D As Object
    Set D = CreateObject("new:{1C3B4210-F441-11CE-B9EA-00AA006B1A69}")
    With D
        .GetFromClipboard
        Debug.Print .GetText
    End With
End Sub
```

如果采用前期绑定的方式，需要事先添加“Microsoft Forms 2.0 Object Library”外部引用。

```
Sub 剪贴板_前期绑定 ()
    Dim D As MSForms.DataObject
    Dim str1 As String, str2 As String
    str1 = "Have a good dream"
    Set D = New MSForms.DataObject
    With D
        .Clear
        .SetText str1
        .PutInClipboard
        .GetFromClipboard
        str2 = .GetText
        Debug.Print str2
    End With
End Sub
```

代码分析：首先把 str1 的内容放到剪贴板，然后从剪贴板获取数据并赋给 str2。

运行上述程序，立即窗口的打印结果仍然是“Have a good dream”。

以上程序的源代码文件为“实例文档 54.xlsm”。

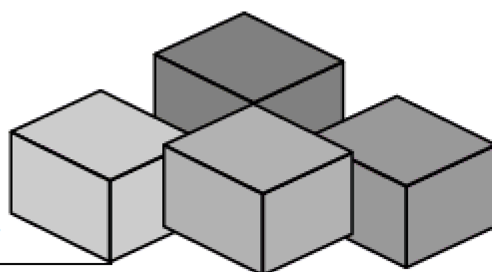
10.3 本章小结

VBIDE 中的 References 集合表示的是引用对话框中处于勾选的那些引用。一个引用的主要属性有 Name、Description、GUID、FullPath 等。

外部对象的 CLSID 和 ProgID 的对应关系可以从注册表中看到。使用 CreateObject 创建新对象，一般使用 ProgID 作为参数。

第 11 章

操作 Acrobat 对象



在日常办公中，PDF 文件的使用频率非常高，而且 Office 文档可以方便地转换为 PDF 文件，如何快速、准确地对 PDF 文件进行自动化处理，也是程序设计中的迫切之需。

Adobe Acrobat 是由 Adobe 公司开发的一款 PDF（(Portable Document Format，便携式文档格式）编辑软件，借助 Acrobat 可以对 PDF 文档进行浏览、打印和编辑，或使用更高级的功能。当计算机中安装了 Adobe Acrobat 之后，系统中会出现一个 Adobe Acrobat X.0 Type Library 的对象库，该对象库也可以被引用到 VBA 工程中，从而实现用 VBA 对 Acrobat 软件及其打开的 PDF 文档进行操作。

本章的主要内容包括 Office 文档自动导出为 PDF 文件，以及使用 VBA 操作 Acrobat 对象，达到用 VBA 自动使用 Acrobat 软件中的命令的目的。

本章用到的外部引用和重要对象：

- Adobe Acrobat 9.0 Type Library
  - Acrobat.AcroApp
  - Acrobat.AcroAVDoc
  - Acrobat.AcroAVPageView
  - Acrobat.AcroPDDoc

11.1 认识 Adobe Acrobat

PDF 文件的浏览、编辑软件非常多，但是 Acrobat 软件是最专业的一款 PDF 软件。Acrobat 软件包括主菜单、工具栏、导航窗格、右键菜单等界面元素，如图 11-1 所示。

其中，主菜单中的【视图】菜单用于设置 PDF 的阅读模式、缩放比例等。【文档】菜单中的命令用于文档编辑，例如增加和删除页面操作等。关于其他菜单，包括工具栏的功能，此处不一一讲解。

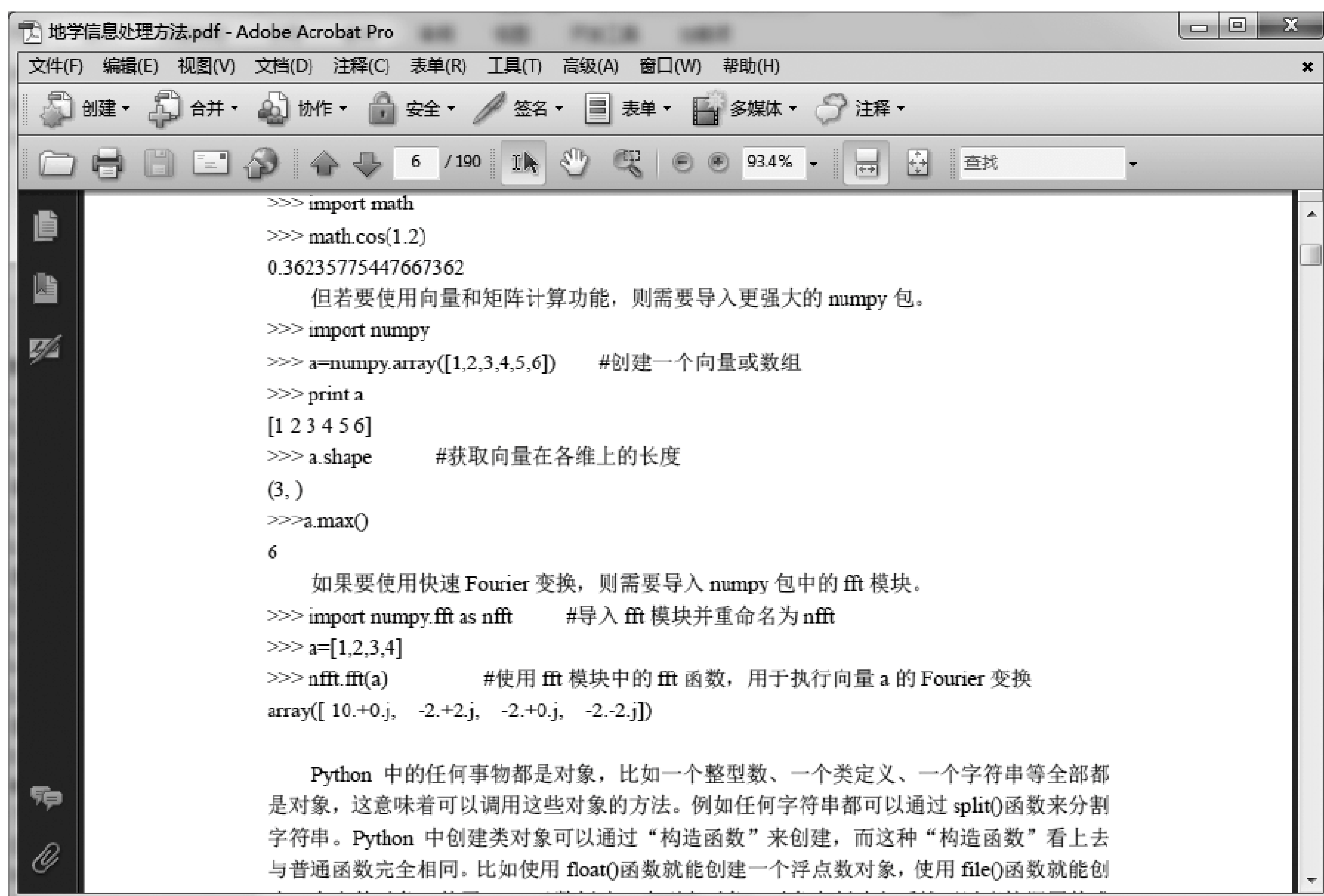


图 11-1 Adobe Acrobat 软件

由于工具栏的数目非常多，通常情况下在屏幕上只显示一部分工具栏，如果要显示或隐藏更多的工具栏，可以单击菜单【工具 / 自定义工具栏】，弹出“更多工具”对话框，如图 11-2 所示。

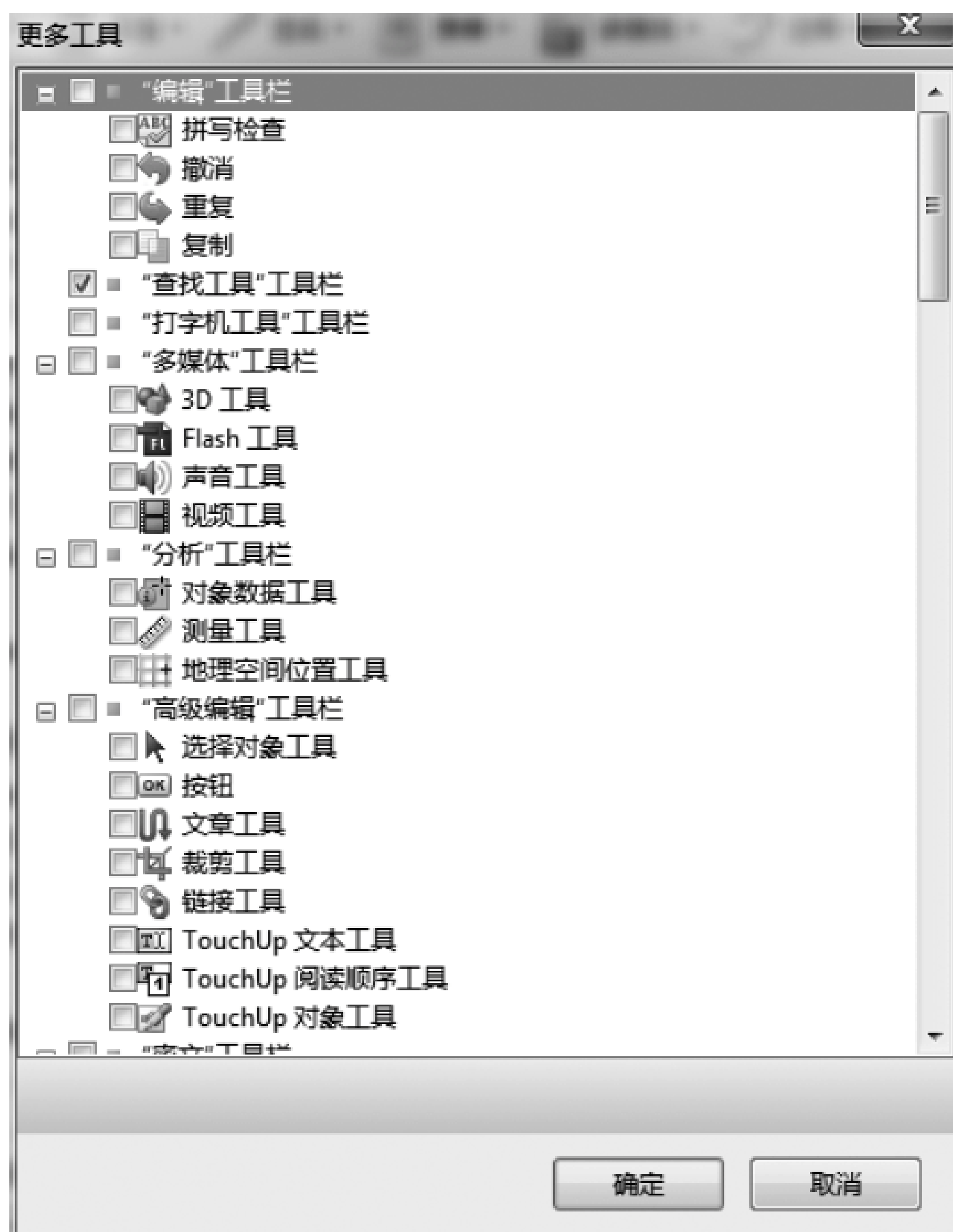


图 11-2 显示更多工具栏

Acrobat 软件在使用方式上比较类似于 Microsoft Word，可以同时打开计算机中多个 PDF 文件，同时该软件的大量常用快捷键也和 Word 中的几乎一样。

11.2　Office 文档导出为 PDF 文件

PDF 文件的生成和创建有非常多的方法，例如用 LaTeX、OpenOffice、微软 Office 都可以生成 PDF 文件。

Office 2013 中的 Excel、PowerPoint、Word 文档均可导出为 PDF 文件，如果手工导出，需要在 backstage 视图中依次单击【导出 / 创建 PDF】，如图 11-3 所示。



图 11-3　Word 文档导出为 PDF 文件

接下来会弹出一个路径选择的对话框，一定要注意对话框右下角有个“选项”按钮，如图 11-4 所示。

“选项”对话框中主要包括导出范围，是全文档导出，还是所选内容导出，如图 11-5 所示。

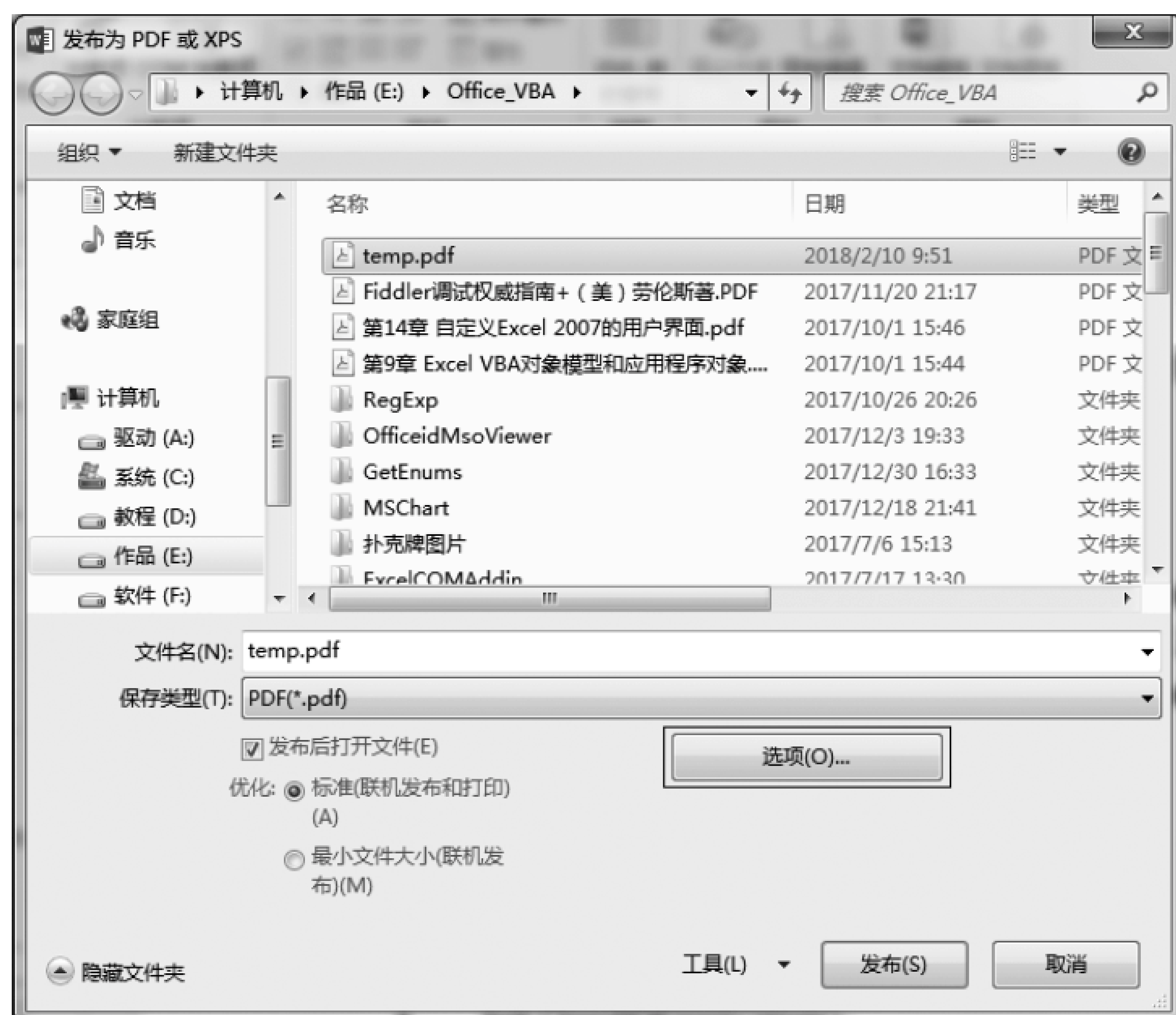


图 11-4　导出选项



图 11-5　“选项”对话框

把以上各项进行恰当设置后，就可以在路径下生成想要的 PDF 文件。

11.2.1 Word 文档导出为 PDF 文件

在 Word VBA 中，Document 对象或 Selection 对象下面的 ExportAsFixedFormat 方法就是用来导出文档的。

编写并运行下面的过程，即可把活动 Word 文档生成为 PDF 文件。

```
Sub 导出文档的全部 ()
     Application.ActiveDocument.ExportAsFixedFormat OutputFileName:="E:\Office_
VBA\temp.pdf", _
    ExportFormat:=wdExportFormatPDF, OpenAfterExport:=True, OptimizeFor:= _
    wdExportOptimizeForPrint, Item:= _
    wdExportDocumentContent, IncludeDocProps:=True, KeepIRM:=True, _
    CreateBookmarks:=wdExportCreateHeadingBookmarks, DocStructureTags:=True, _
    BitmapMissingFonts:=True, UseISO19005_1:=False
End Sub
```

ExportAsFixedFormat 方法的各个参数说明如下。

OutputFileName:="E:\Office_VBA\temp.pdf" 表示导出文件的目标名称和路径。

ExportFormat:=wdExportFormatPDF 表示导出的格式是 PDF 文件。

OpenAfterExport:=True 表示导出操作完成后自动打开 PDF 文件。

OptimizeFor:=wdExportOptimizeForPrint 表示“优化（标准 联机发布和打印）”。

Item:=wdExportDocumentContent 表示导出文档所有内容。

IncludeDocProps:=True 表示勾选“文档属性”。

KeepIRM:=True 未知。

CreateBookmarks:=wdExportCreateHeadingBookmarks 表示勾选“创建书签时使用标题”。

DocStructureTags:=True 表示勾选“辅助功能文档结构标记”。

BitmapMissingFonts:=True 表示勾选“无法嵌入字体情况下使用文本位图”。

UseISO19005_1:=False 表示不勾选“符合 ISO19005-1 标准”。

如果要导出 Word 文档中鼠标选中的内容，只需要把上述代码中的 ActiveDocument 换成 Selection 即可。

```
Sub 导出所选内容 ()
    Application.Selection.ExportAsFixedFormat OutputFileName:="E:\Office_VBA\
temp.pdf", _
    ExportFormat:=wdExportFormatPDF, OpenAfterExport:=True, OptimizeFor:= _
    wdExportOptimizeForPrint, ExportCurrentPage:=False, Item:= _
    wdExportDocumentContent, IncludeDocProps:=True, KeepIRM:=True, _
    CreateBookmarks:=wdExportCreateHeadingBookmarks, DocStructureTags:=True, _
    BitmapMissingFonts:=True, UseISO19005_1:=False
End Sub
```

11.2.2 Excel 工作簿导出为 PDF 文件

Excel 导出为 PDF 的选项对话框，内容略有不同，如图 11-6 所示。

Excel VBA 中的 Workbook、Worksheet、Range 对象之后都有 ExportAsFixedFormat 方法，分别表示把整个工作簿、工作表、单元格导出为 PDF 文件。

例如：

ActiveWorkbook. ExportAsFixedFormat …表示把活动工作簿导出为 PDF。

Worksheets(2). ExportAsFixedFormat …表示把第二个工作表导出为 PDF。

Application.ActiveSheet. ExportAsFixedFormat …表示把活动工作表导出为 PDF。

Application.Selection. ExportAsFixedFormat …表示把鼠标所选单元格区域导出为 PDF。



图 11-6　Excel 工作簿导出为 PDF 的选项对话框

下面的代码把 Excel 所选区域导出为 PDF 文件。

```
Sub Excel 导出为 PDF()
    Selection.ExportAsFixedFormat Type:=xlTypePDF, Filename:= _
    "E:\temp.pdf", Quality:= _
    xlQualityStandard, IncludeDocProperties:=False, IgnorePrintAreas:=False, _
    OpenAfterPublish:=True
End Sub
```

11.2.3　PowerPoint 演示文稿导出为 PDF 文件

PowerPoint 导出 PDF 的选项对话框如图 11-7 所示。

下面的过程把活动演示文稿导出为 PDF 文件。

```
Sub 演示文稿导出为 PDF()
    Application.ActivePresentation.ExportAsFixed Format
Path:="C:\temp\temp.pdf", FixedFormatType:=PowerPoint.
PpFixedFormatType.ppFixedFormatTypePDF
End Sub
```

关于 ExportAsFixedFormat 方法更多的参数说明，请参阅微软提供的 MSDN 在线帮助。

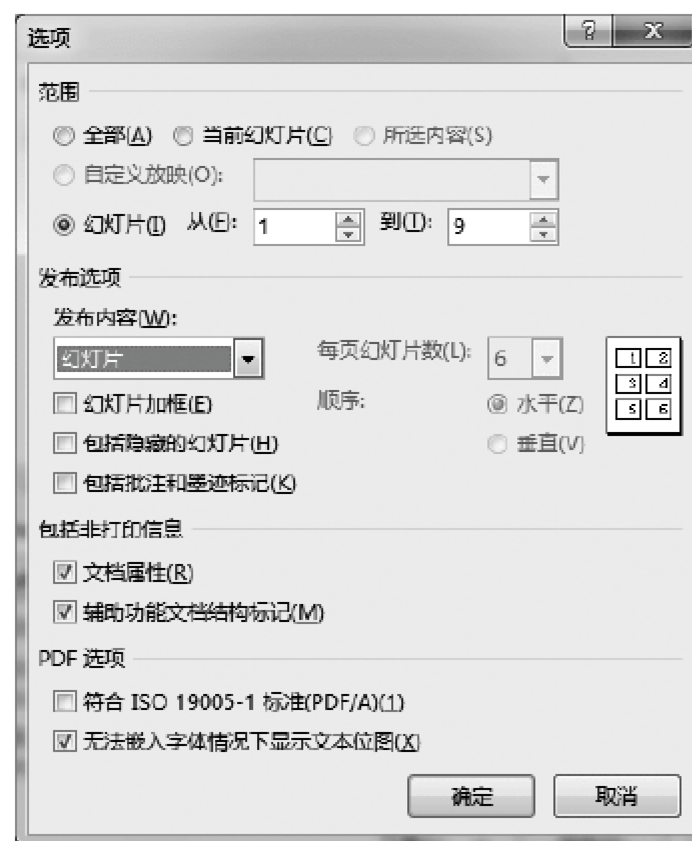


图 11-7　PowerPoint 文件导出为 PDF 的选项对话框

11.3　Acrobat 对象模型

计算机中安装了 Adobe Acrobat 软件后，可以用 VBA 访问 Acrobat 软件以及 PDF 文档。

11.3.1　引用 Acrobat 对象库

VBA 中访问 Acrobat 对象，需要事先为 VBA 工程添加“Adobe Acrobat X.0 Type Library”引用，如图 11-8所示。

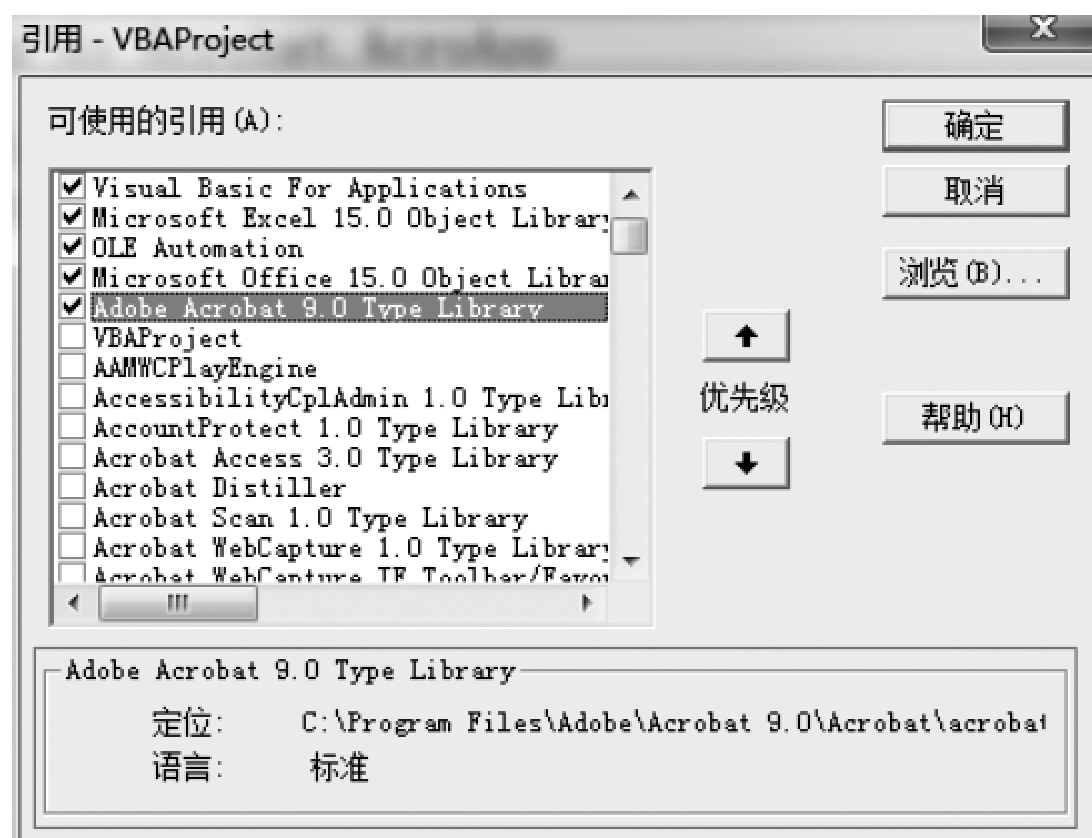


图 11-8　添加外部引用

当在 VBA 工程中引用了 Acrobat，按下 F2 键后在对象浏览器中可以看到 Acrobat 的对象模型，如图 11-9 所示。

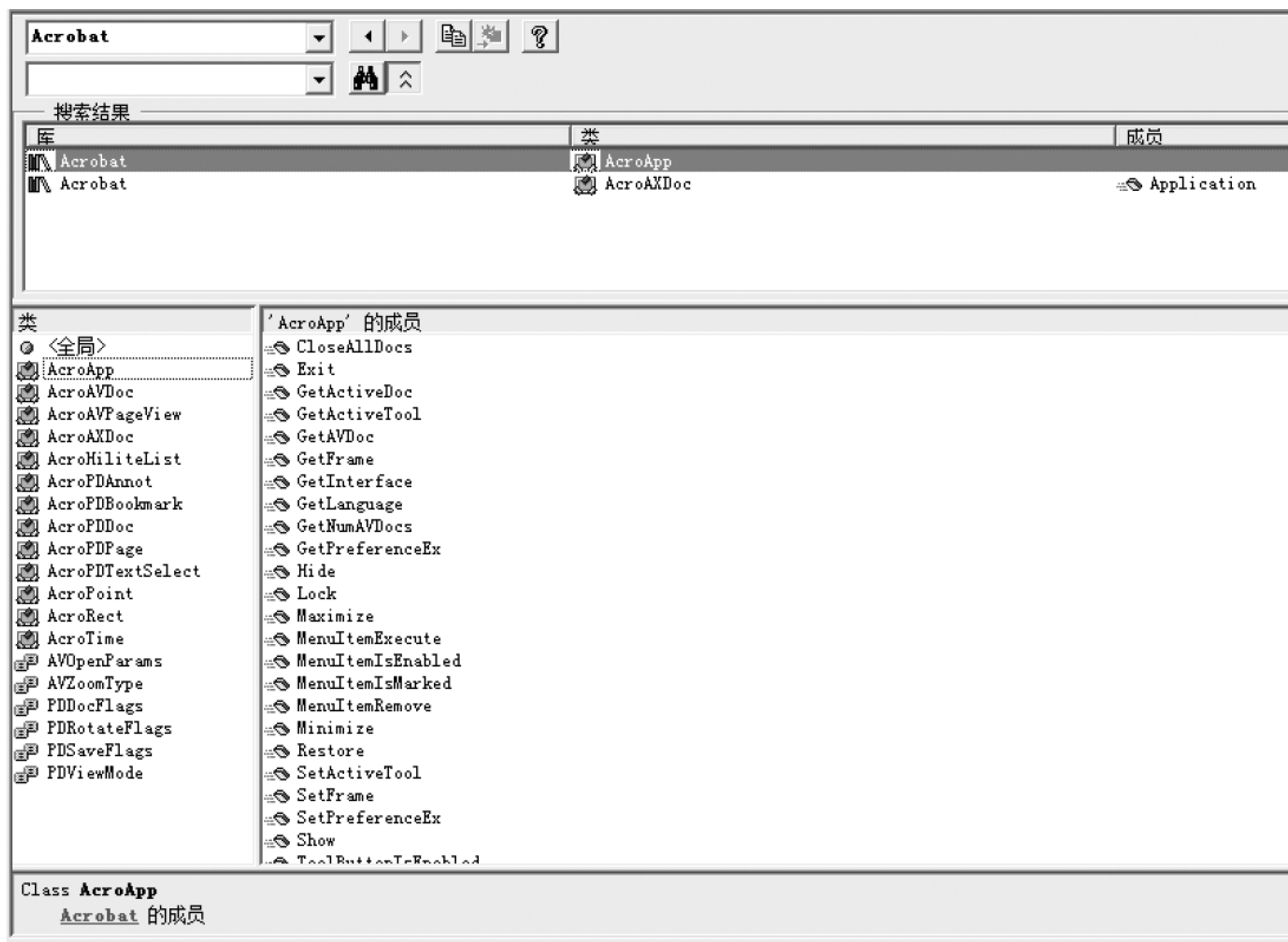


图 11-9　Acrobat 对象库的主要成员

11.3.2　Acrobat 常用对象

利用 VBA 操作 Acrobat 软件以及在 Acrobat 软件中打开的 PDF 文档，需要了解 AcroApp、AcroAVDoc、AcroPageView 对象，如果要直接操作计算机中的 PDF 文件，可以

使用 AcroPDDoc 对象后台方式打开文档，无论哪一种方式，都可以用 AcroPDPage 对象来操作 PDF 文档的页面。

Acrobat 常用的对象模型如图 11-10 所示。

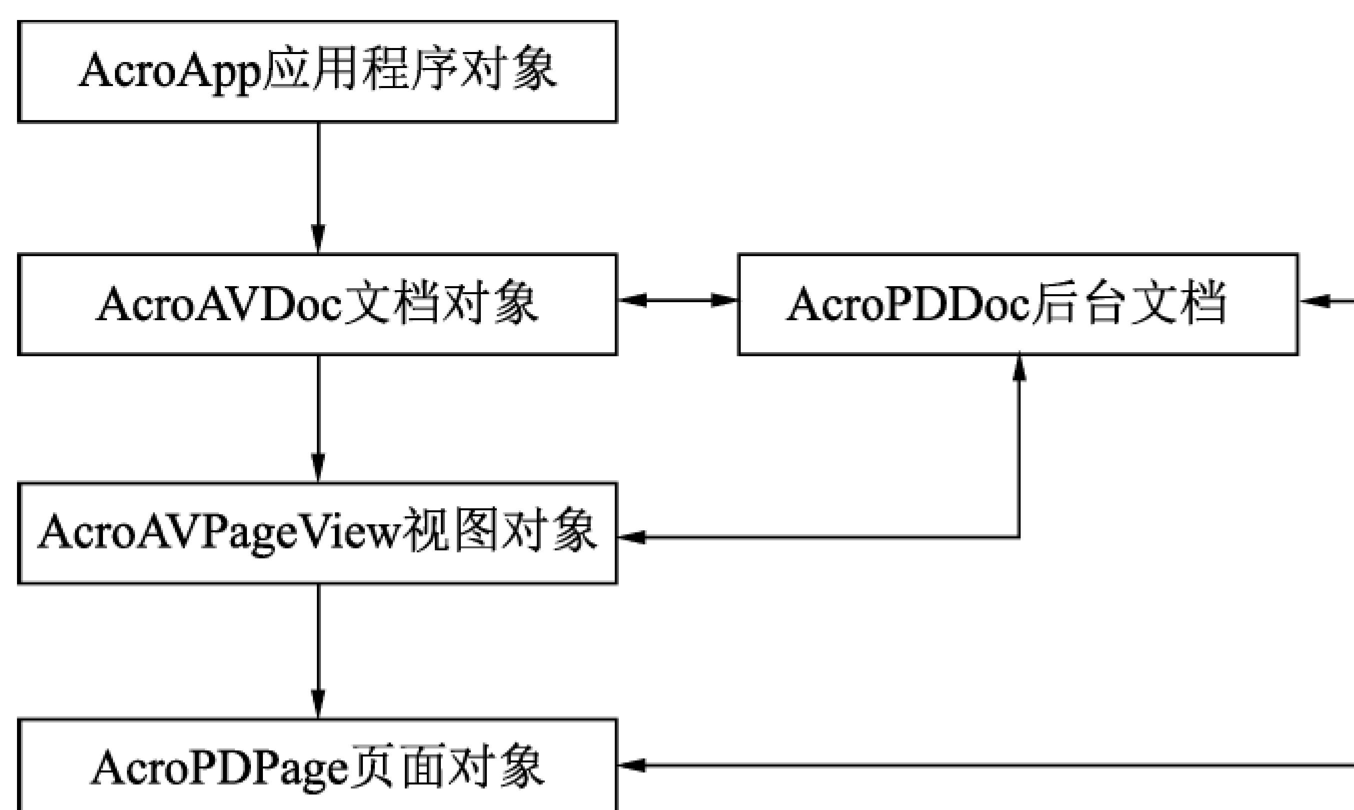


图 11-10　Acrobat 常用对象模型

如果图 11-10 中的箭头是双向箭头，表示对象之间可以互相获得。

11.3.3　Acrobat 枚举常量

当 VBA 工程中引入了 Acrobat 对象库，除了可以声明相关的对象类型，还会用到对象库中的内置枚举常量，这些常量都以 Acrobat 开头。

例如下面的语句是把 Acrobat 中的当前 PDF 文档页面缩放为“适合页面宽度”。

```
App.GetActiveDoc.GetAVPageView.ZoomTo Acrobat.AVZoomType.AVZoomFitWidth,100
```

运行上述代码，会自动勾选 Acrobat 中的菜单项【视图 / 缩放 / 适合宽度】，如图 11-11 所示。

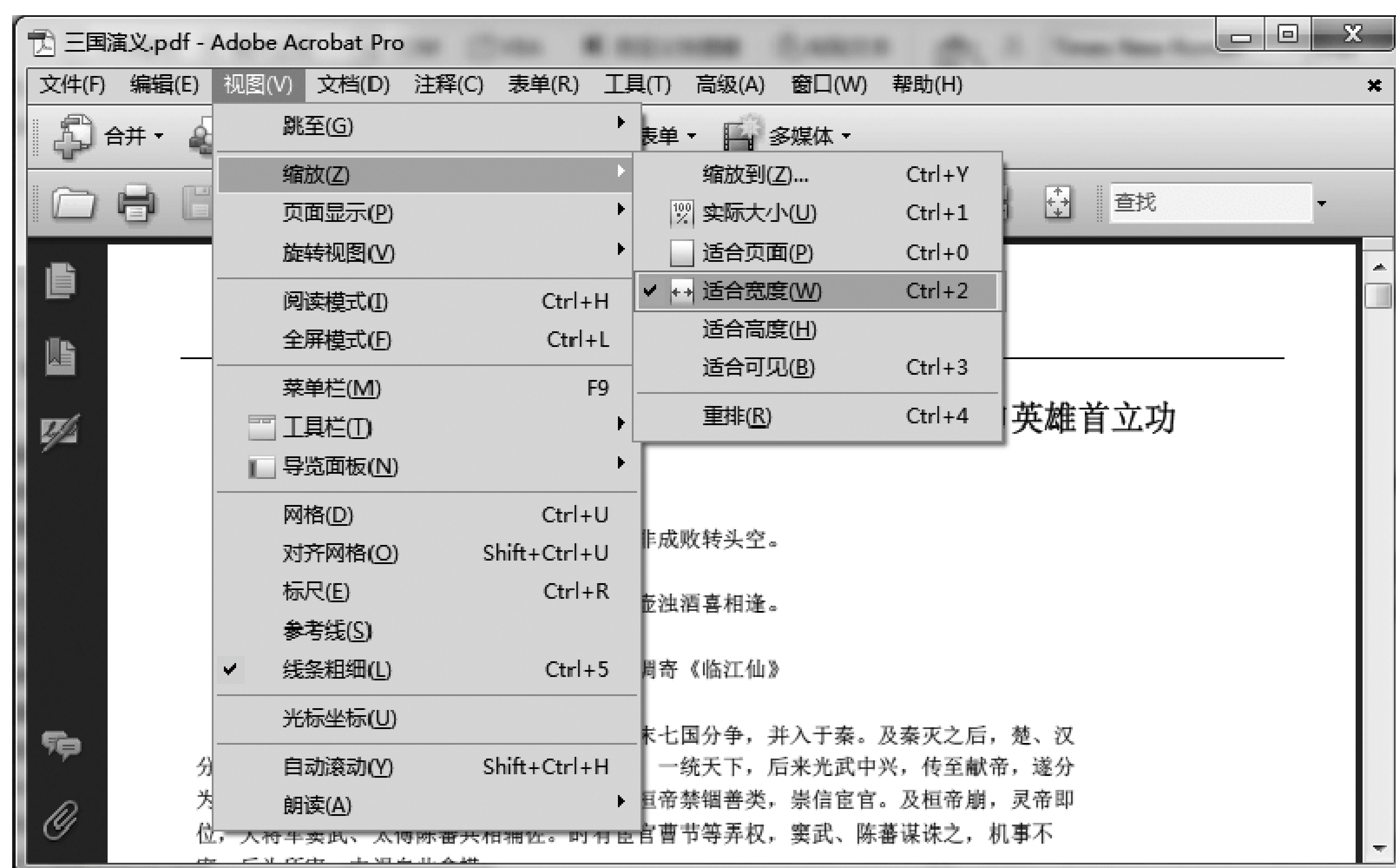


图 11-11　使用 VBA 自动更改缩放比例

11.4 AcroApp 应用程序对象

Acrobat 的 AcroApp 对象类似于 Excel VBA 中的 Application 对象，表示一个打开的 Acrobat 应用程序。AcroApp 对象有大量属性和方法，如表 11-1 所示。

表 11-1 AcroApp 对象的常用属性和方法

属性或方法名称	功　能
CloseAllDocs	关闭所有打开的 PDF 文档
GetActiveDoc	返回活动 PDF 文档对象
GetActiveTool	返回正在使用的“工具”
GetAVDoc(i)	返回打开的第 i 个 PDF 文档
GetFrame	返回应用程序窗口所在的矩形，Rect 对象
GetNumAVDocs	返回打开的 PDF 文档总数
Hide	隐藏 Acrobat 软件
MenuItemExecute	自动执行工具栏控件命令
SetActiveTool	自动设置活动“工具”
SetFrame	重新规定应用程序窗口的位置
Show	显示 Acrobat 软件

11.4.1 创建 Acrobat 对象

下面的过程打开 Acrobat 软件，显示 / 隐藏 Acrobat 窗口。由于 Exit 方法不好用，因此使用 Shell 调用任务管理器，结束 Acrobat 进程的方式退出 Acrobat 软件。

```
Public App As Acrobat.AcroApp
Sub Test1()
    Set App = CreateObject("AcroExch.App")
    With App
        .Show
        .Hide
        Shell "taskkill /f /im acrobat.exe", vbHide
    End With
End Sub
```

11.4.2 获取已经打开的 Acrobat 对象

下面的过程通过 GetObject 获取屏幕上已经打开的 Acrobat 软件，如果事先用该软件已经打开了一些 PDF 文档，运行下面的过程将弹出打开的文档数目。

```
Sub Test2()
    Set App = GetObject("", "AcroExch.App")
    With App
        MsgBox "打开的文档数目：" & .GetNumAVDocs
    End With
End Sub
```

11.4.3　获取和设置活动工具

在浏览 PDF 文档的时候，鼠标的默认指针是一个左上方向的箭头，如果单击工具栏或菜单中的“手形工具”，鼠标光标会变为手的样子，如图 11-12 所示。

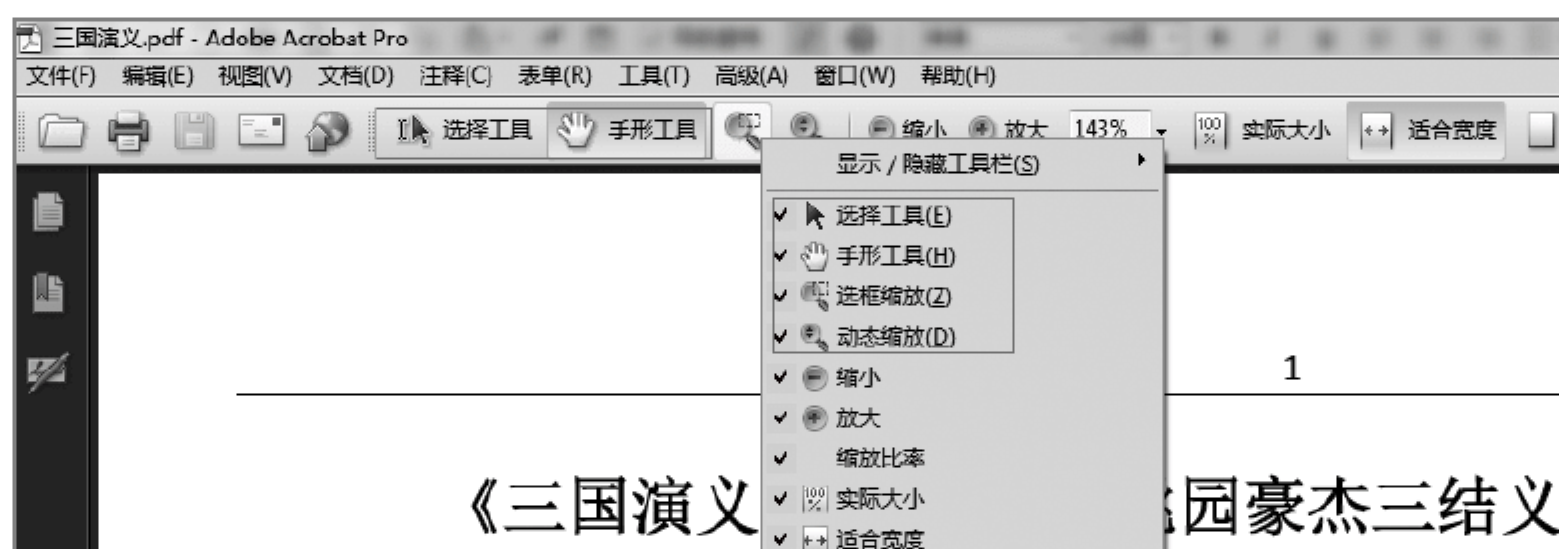


图 11-12　Acrobat 中的工具栏

这就涉及活动工具的状态获取和设置。

AcroApp 对象的 GetActiveTool 方法根据 Acrobat 软件目前的状态返回一个用字符串表达的当前活动工具。常用的活动工具名称如下。

- ❑ Hand
- ❑ Note
- ❑ Select
- ❑ SelectGraphics
- ❑ Zoom
- ❑ Link
- ❑ Thread

下面的代码打印当前活动工具的名称，并且自动设置活动工具为“手形”。

```
Sub 活动工具的获取和设置 ()
    Set App = GetObject("", "AcroExch.App")
    With App
        Debug.Print " 当前活动工具是: ", .GetActiveTool
        .SetActiveTool "Hand", True
    End With
End Sub
```

11.4.4　自动执行 Acrobat 工具栏控件命令

AcroApp 对象的 MenuItemExecute 方法可以自动执行 Acrobat 中的一个工具栏命令，下面的代码自动设置为“适合页面”，并且跳转到最后一页。

```
Sub 执行工具栏命令 ()
    Set App = GetObject("", "AcroExch.App")
    With App
        .MenuItemExecute "FitPage"
        .MenuItemExecute "LastPage"
```

```
    End With
End Sub
```

代码分析：MenuItemExecute "LastPage" 相当于单击了 Acrobat 菜单项中的【视图 / 跳至 / 最后一页】，如图 11-13 所示。

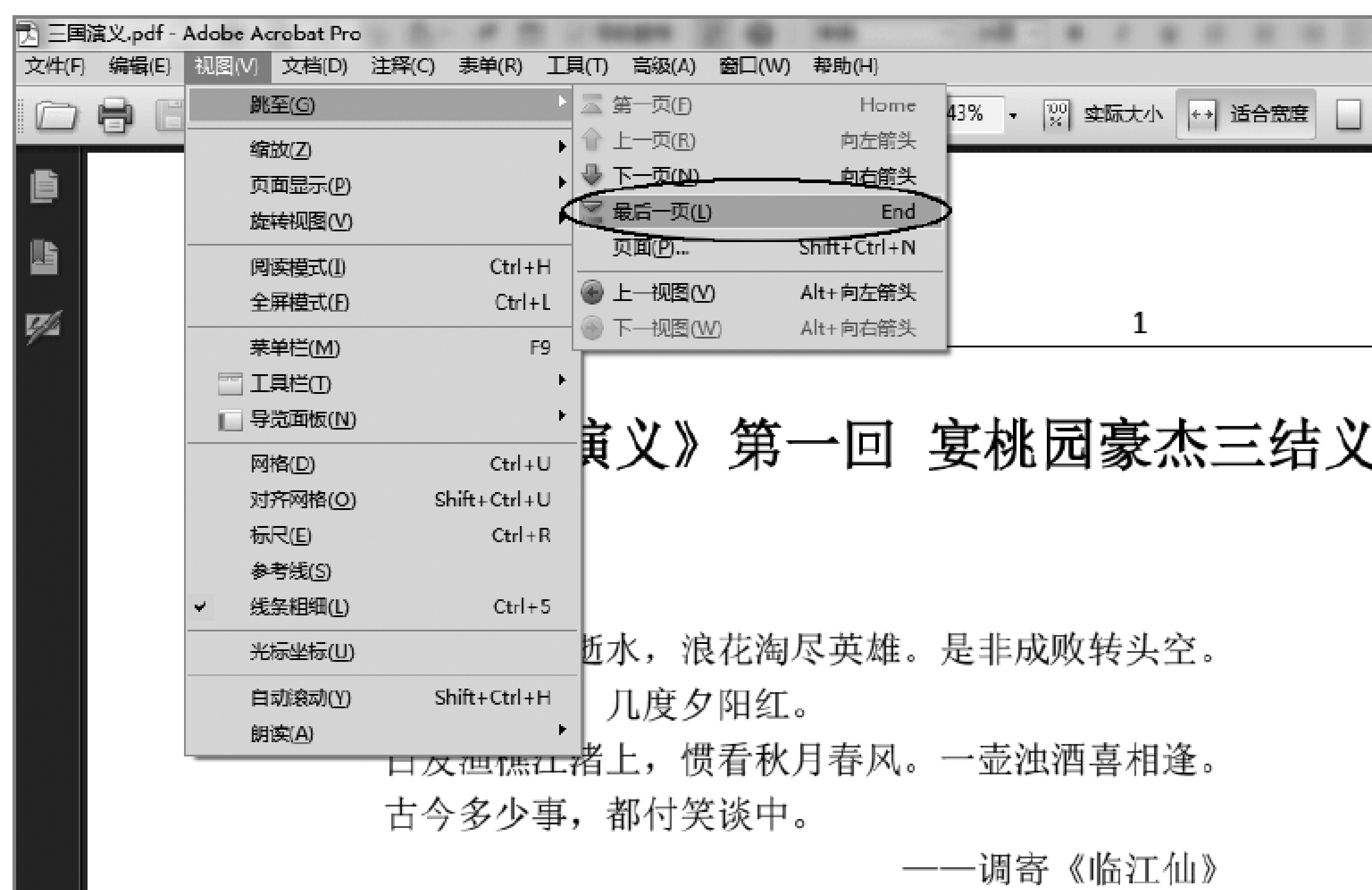


图 11-13　跳转到最后一页

Acrobat 对象中常用工具栏控件的名称常量如表 11-2 所示。

表 11-2　Acrobat 对象中常用工具栏控件的名称常量

常　量	功　能	常　量	功　能
Hand	手形工具	FirstPage	第一页
ZoomIn	缩小	PreviousPage	上一页
ZoomOut	扩大	NextPage	下一页
Select	选择	LastPage	最后一页
GoBack	返回上一视图	ShowHideToolBar	显示 / 隐藏工具栏
GoForward	跳转下一视图	ShowHideMenuBar	显示 / 隐藏菜单栏
Zoom100	显示整页	Cascade	窗口 / 层叠
FitPage	适合页面	TileHorizontal	水平平铺
FitVisible	适合可见	TileVertical	垂直平铺

11.5　AcroAVDOC 文档对象

AcroApp 对象的 GetNumAVDocs 属性返回打开的文档数目，GetAVDoc(i) 用来获取第 i 个 PDF 文档，当 i 是 0 时表示第一个 PDF 文档，返回一个 AcroAVDoc 对象。特别地，AcroApp 的 GetActiveDoc 直接获取到当前活动 PDF 文档。

11.5.1　遍历所有打开的 PDF 文档

下面的代码遍历当前打开的所有 PDF 文档，然后打印每个文档的标题。

```
Sub 遍历打开的 PDF 文件 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim i As Integer
    Set App = GetObject("", "AcroExch.App")
    For i = 0 To App.GetNumAVDocs - 1
        Set Doc = App.GetAVDoc(i)
        With Doc
            .BringToFront
            Debug.Print "标题：", .GetTitle
        End With
    Next i
End Sub
```

代码中的 BringToFront 表示把一个文档前置，成为活动文档。

运行上述过程，立即窗口的打印结果如图 11-14 所示。

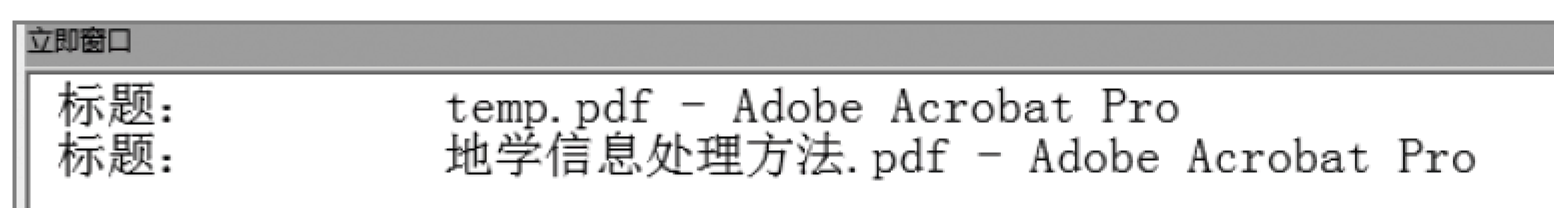


图 11-14　遍历所有打开的 PDF 文档

11.5.2　AcroAVDOC 对象的属性和方法

AcroAVDOC 对象的属性和方法如表 11-3 所示。

表 11-3　AcroAVDOC 对象的属性和方法

属性或方法名称	功　能
BringToFront	把文档前置
ClearSelection/ShowTextSelect	清除选择 / 显示选择
Open/Close	打开 / 关闭文档
FindNext	查找下一处
GetAVPageView	获取 Page 视图
GetPDDoc	获取 AcroPDDoc 对象
GetTitle/SetTitle	获取 / 设置窗口标题
GetViewMode/SetViewMode	获取 / 设置视图模式
IsValid	是否有效的文档
GetFrame/SetFrame	获取 / 设置 PDF 文档窗口的位置
PrintPages	打印指定的页码范围

下面的过程使用 AcroAVDoc 的 Open 方法打开计算机中的一个 PDF 文档。

```
Sub 打开 PDF 文件 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc, Result As Boolean
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = GetObject("", "AcroExch.AVDoc")
```

```
    With Doc
        Result = .Open("C:\temp\temp.pdf", "")
        If Result Or Doc.IsValid Then
            MsgBox "成功打开！", vbInformation
        End If
        .BringToFront
        .Maximize True
    End With
End Sub
```

代码分析：Set Doc = GetObject("", "AcroExch.AVDoc") 这句，也可以换成 Set Doc = CreateObject ("AcroExch.AVDoc")，用来创建一个新的 AcroAVDoc 对象。

执行 Open 方法后，返回一个布尔值，如果打开成功，则 Result 是 True，同时 IsValid 属性也是 True。

关闭 PDF 文件，可以使用 AcroAVDoc 的 Close 方法关闭，也可以使用 AcroApp 的 CloseAllDocs 关闭所有文件。

假设现在 Acrobat 软件中打开了一个以上的 PDF 文件，执行如下过程，首先关闭当前活动 PDF 文件，然后关闭所有文件。

```
Sub 关闭 PDF 文件 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc, Result As Boolean
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Doc.Close bNoSave:=True
    App.CloseAllDocs
End Sub
```

代码分析：Doc.Close bNoSave:=True 表示关闭时不保存修改。

11.5.3 清除选择和显示选择

在浏览 PDF 文件的时候，用鼠标选中一个内容区域后，可能翻页到其他页面，此时使用 AcroAVDoc 的 ShowTextSelect 方法可以快速跳转到鼠标选中的区域。

同时，使用 ClearSelection 还可以取消选择。

```
Sub 清除选择与显示选择 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Doc.ShowTextSelect
    MsgBox "下面将自动清除选择。"
    Doc.ClearSelection
End Sub
```

11.5.4　在 PDF 文件中查找内容

使用 AcroAVDoc 对象的 FindNext 方法，可以实现自动在 PDF 文件中查找指定的文字，如果找到，返回 True，并且在 PDF 文件中自动跳转到目标位置。

FindNext 方法有 4 个需要指定的参数：查找关键词、是否区分大小写、是否整词匹配、是否从头查找。

下面的过程在当前活动 PDF 文档中查找“Python”。

```
Sub 查找内容 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim result As Boolean, i As Integer
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    For i = 1 To 10
        result = Doc.FindText(szText:="Python", bCaseSensitive:=True, bWholeWordsOnly:
=False, bReset:=False)
        If result = False Then Exit For
    Next i
End Sub
```

代码分析：上述实例中，设置为查找前 10 个关键词，每查找到一个，在 PDF 文件中会自动选中并跳转到目标位置。如果参数 bReset 是 True，则每次都从文档开头处查找，查到的 10 个都是头一个位置。

11.5.5　获取和设置 PDF 标题文字

AcroAVDoc 对象的 GetTitle 和 SetTitle 用来获取和设置 PDF 文件的 Title 属性。

```
Sub 获取和设置标题文字 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim result As Boolean, i As Integer
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Debug.Print Doc.GetTitle
    Doc.SetTitle " 开心一刻 "
    Debug.Print Doc.GetTitle
End Sub
```

运行上述代码后，Acrobat 的左上角标题被更改，如图 11-15 所示。



图 11-15　获取和修改标题

11.5.6 获取和设置阅览模式

AcroAVDoc 的 GetViewMode 和 SetViewMode 用来获取和设置阅览视图模式。

运行下面的过程，使得 PDF 文件进入全屏预览模式。

```
Sub 获取和设置阅览模式 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim result As Boolean, i As Integer
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Debug.Print Doc.GetViewMode
    Doc.SetViewMode nType:=Acrobat.PDViewMode.PDFullScreen
End Sub
```

代码分析：SetViewMode 的可用参数来自 Acrobat.PDViewMode 的枚举值。

11.5.7 获取和设置 PDF 文档窗口位置

AcroAVDOC 对象的 GetFrame 返回一个 AcroRect 对象，该对象具有 Left、Top、Right 和 Bottom 四个属性，用于返回 PDF 文档窗口的左上角、右下角在屏幕上的坐标。

```
Sub 获取和设置 PDF 文档窗口位置 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim result As Boolean, i As Integer
    Dim Rect As Acrobat.AcroRect
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Set Rect = Doc.GetFrame
    With Rect
        Debug.Print .Left, .Top, .Right; .bottom
    End With
    MsgBox "接下来自动设置 Acrobat 窗口位置！"
    With Rect
        .Left = 200
        .Top = 220
        .Right = 1000
        .bottom = 600
    End With
    Doc.SetFrame Rect
End Sub
```

运行以上代码，首先打印出当前 PDF 文档窗口的所在位置，然后重设位置。

11.5.8 打印或另存 PDF 文档

AcroAVDoc 对象的 PrintPages 方法会在 Acrobat 中自动弹出打印对话框，如果计算机中没有找到可用的打印机，则另存为 PDF 文档。

下面的代码打印活动文档的第 2 ～ 5 页。

```
Sub 打印 PDF 文档 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim result As Boolean, i As Integer
    Dim Rect As Acrobat.AcroRect
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Doc.PrintPages nFirstPage:=2, nLastPage:=5, nPSLevel:=2, bBinaryOK:=False,
bShrinkToFit:=False
End Sub
```

以上内容的源代码文件为“实例文档 71.xlsm”。

AcroAVDoc 对象的 GetPDDoc 属性用来获取一个 AcroPDDoc 对象，GetAVPageView 属性用来获取一个 AcroAVPageView 对象，下面分别介绍。

11.6　AcroAVPageView 对象

AcroAVPageView 对象可以对 Acrobat 文档的视图和浏览方式进行访问和设定，比较常用的方法和属性如下。

- GetPageNum/GoTo，获取文档当前页码 / 跳转到某页。
- GetZoom/ZoomTo，获取 / 设置当前缩放比例。
- ReadPageDown/ReadPageUp，跳转到下一页 / 上一页。

下面的代码获取文档当前页码并自动跳转到指定的页面。然后更改视图缩放比例。

```
Sub 更改 PDF 文档视图 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim APV As Acrobat.AcroAVPageView
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Set APV = Doc.GetAVPageView
    With APV
        Debug.Print "当前页码：", .GetPageNum
        .GoTo 6
        Debug.Print "当前缩放比例：", .GetZoom
        .ZoomTo Acrobat.AVZoomType.AVZoomNoVary, 75     '缩放到 75%
        .ReadPageDown                                   '按下 Page Down 键
        .ReadPageUp                                     '按下 Page Up 键
    End With
End Sub
```

运行上述代码后，可以看到自动跳转到第 7 页，并且缩放比例为 75%，如图 11-16 所示。

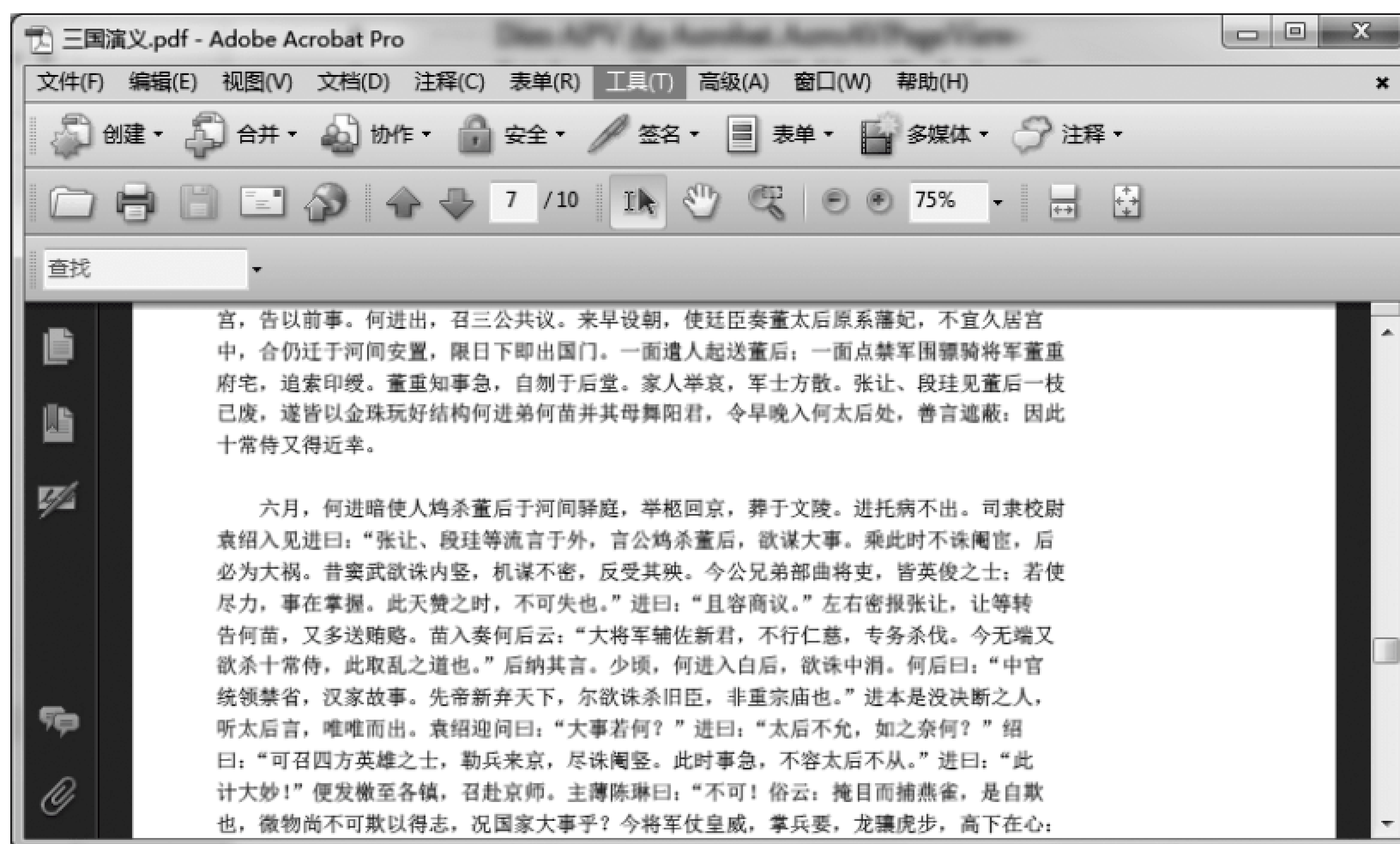


图 11-16 自动跳转和调整缩放比例

AcroAVPageView 对象的 GetPage 属性返回一个 AcroPDPage 对象。

11.7 AcroPDPage 对象

AcroPDPage 对象是一个针对“页”操作的对象，也就是 PDF 文档中的某一页进行操作。

11.7.1 获取和更改 PDF 页面旋转角度

下面的代码首先把活动 PDF 文档自动跳转到第 4 页，然后把当前页赋给变量 PDPage。

```
Sub 更改 PDF 页面旋转角度 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim APV As Acrobat.AcroAVPageView
    Dim PDPage As Acrobat.AcroPDPage
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Set APV = Doc.GetAVPageView
    APV.GoTo 4
    Set PDPage = APV.GetPage
    With PDPage
        Debug.Print " 当前页码: ", .GetNumber
        Debug.Print " 当前页面旋转角度: ", .GetRotate
        .SetRotate 180
    End With
End Sub
```

代码分析：SetRotate 方法可用的参数只能是 0、90、180、270 这四个角度值。以上代码执行后，PDF 文档的第 4 页倒立了起来，如图 11-17 所示。

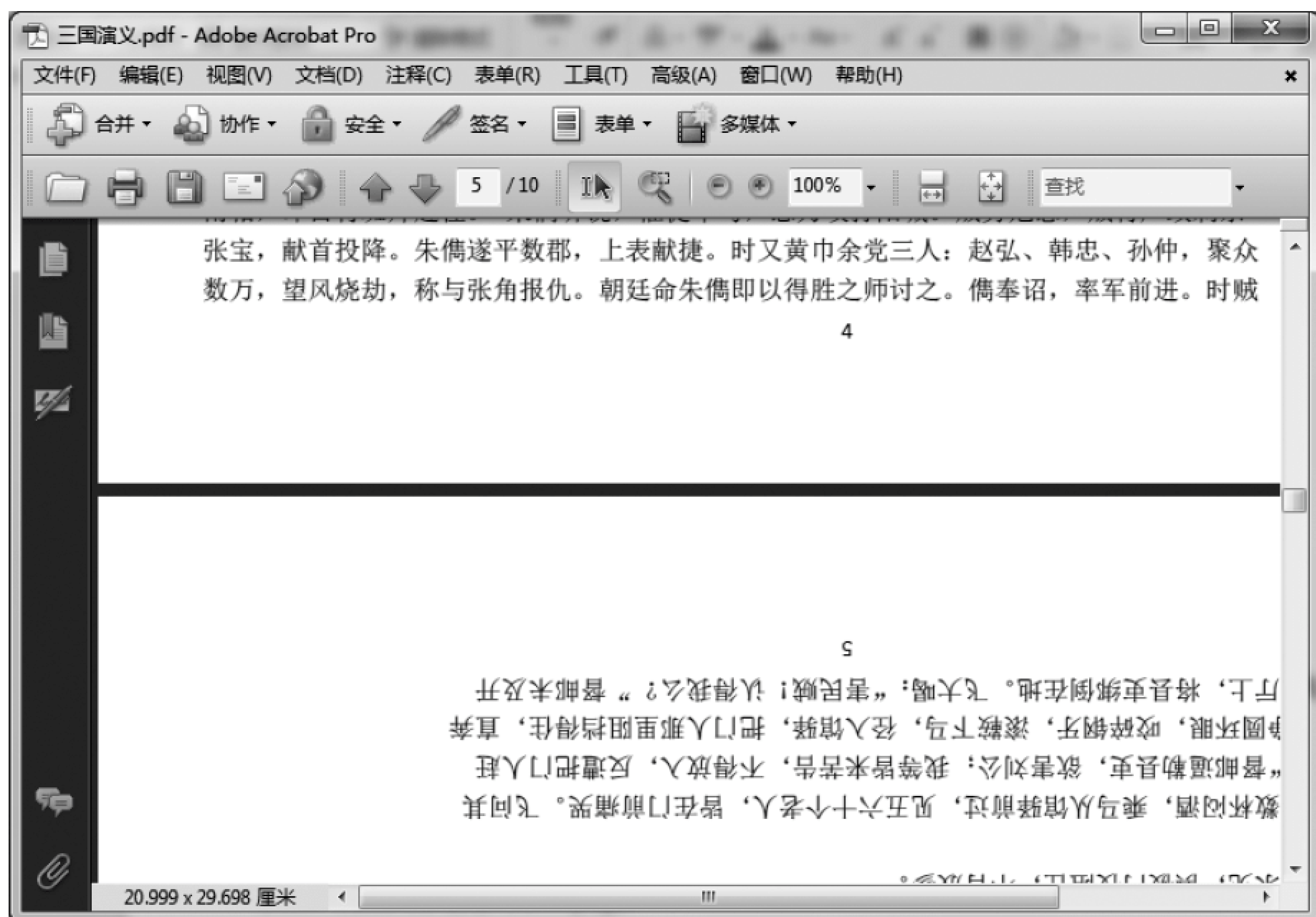


图 11-17　自动旋转页面

11.7.2　删除注释

通过 AcroPDPage 对象，可以对当前页面增加和删除注释。注释是 PDF 文档中的一种元素，类似于 Office 中的批注功能。

下面的代码删除活动 PDF 文档第 4 页中的所有注释。

```
Sub 删除注释 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim APV As Acrobat.AcroAVPageView
    Dim PDPage As Acrobat.AcroPDPage, i As Integer, Count As Integer
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Set APV = Doc.GetAVPageView
    APV.GoTo 4
    Set PDPage = APV.GetPage
    With PDPage
        Count = .GetNumAnnots
        For i = 1 To Count
            .RemoveAnnot 0
        Next i
    End With
End Sub
```

代码分析：GetNumAnnots 属性返回注释的个数，然后删除所有注释。

假设代码运行前，页面如图 11-18 所示。运行上述过程后，页面中的 3 个形状被移除。

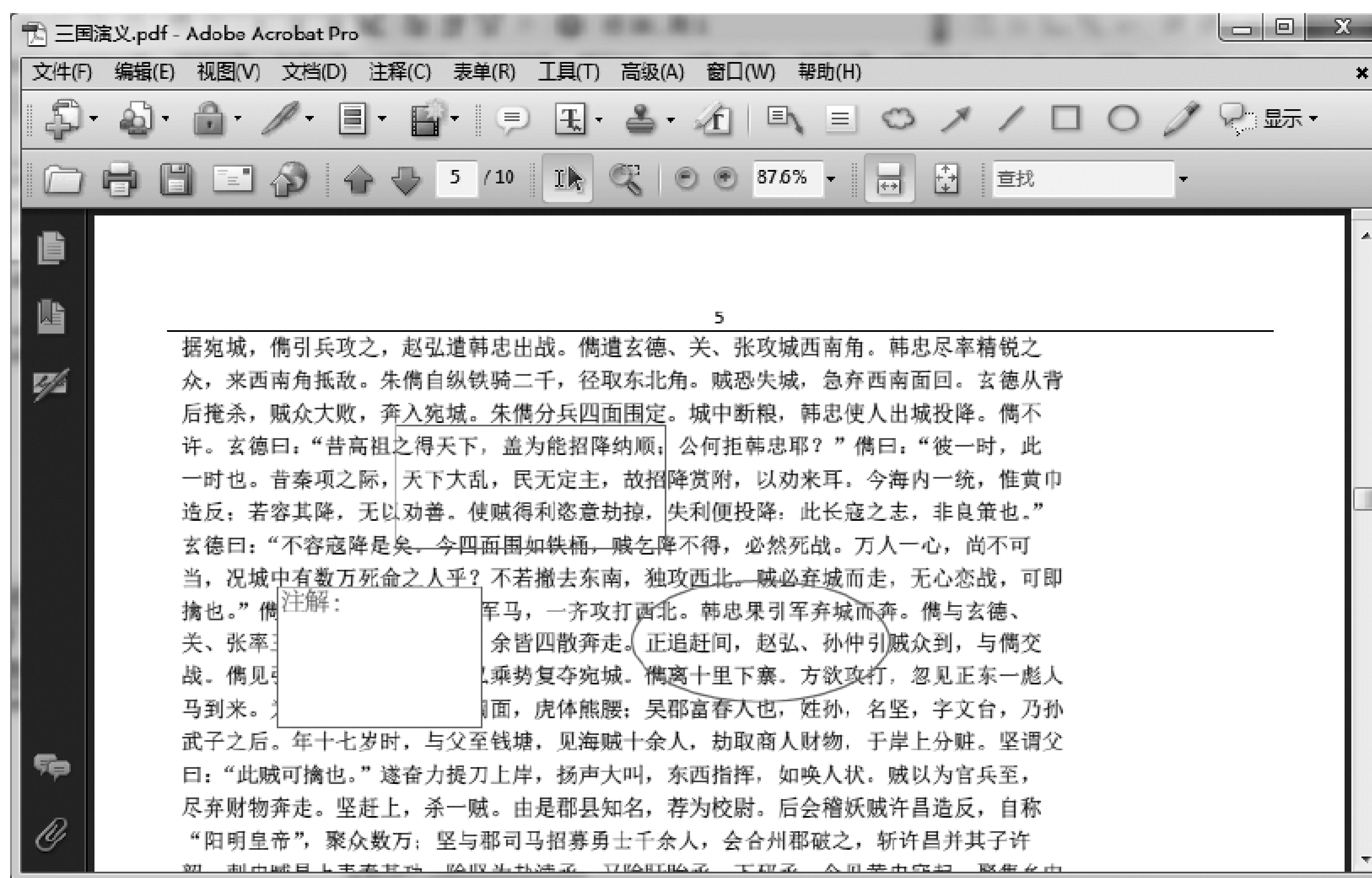


图 11-18　自动删除所有注释

11.7.3　提取页面文字

AcroPDPage 对象联合其他对象，可以遍历页面上的每个字符，从而实现 PDF 文档文字提取的功能。

```
Sub 提取页面文字 ()
    Dim App As Acrobat.AcroApp
    Dim Doc As Acrobat.AcroAVDoc
    Dim APV As Acrobat.AcroAVPageView
    Dim PDPage As Acrobat.AcroPDPage
    Dim AHL As Acrobat.AcroHiliteList
    Dim PDTS As Acrobat.AcroPDTextSelect
    Dim i As Long, S As String
    Set App = GetObject("", "AcroExch.App")
    App.Show
    Set Doc = App.GetActiveDoc
    Set APV = Doc.GetAVPageView
    APV.GoTo 4
    Set PDPage = APV.GetPage
    With PDPage
        Set AHL = New AcroHiliteList
        AHL.Add 0, 32767
        Set PDTS = .CreateWordHilite(AHL)
        With PDTS
            For i = 0 To .GetNumText - 1
                S = S & .GetText(i)
            Next i
        End With
    End With
    Debug.Print S
End Sub
```

代码分析：上述代码自动跳转到活动 PDF 文档的第 4 页，把遍历到的字符连接到字符串变量 S 中，最后输出 S，就是该页的所有文本内容。

以上实例只提取了一页的内容，如果要提取 PDF 文档所有页面内容，只需要在外层再套一个循环页码，就可以实现，如果把输出的字符串发送到 Excel 或 Word，就实现了 PDF 转 Office 文档的功能。

如果要在不打开 PDF 文档的前提下实现转换，可以用 AcroPDDoc 对象在后台打开 PDF 文档，然后使用 AcroPDDoc.AcquirePage(i) 的方式返回第 i 页的 AcroPDPage 对象即可。

以上内容的源代码文件为“实例文档 73.xlsm”。

11.8　AcroPDDoc 对象

AcroPDDoc 对象可以在后台操作 PDF 文件，也就是不需要 AcroApp 对象。

AcroPDDoc 对象与 AcroAVDoc 比较，包含的属性和方法有很多不同，例如 AcroPDDoc 对象可以对 PDF 文件中的页面进行增加、删除和替换操作。

11.8.1　获取和修改 PDF 文件属性

下面的过程获取一个 PDF 文件的属性，实现流程是：打开→获取→关闭。

```
Sub 后台获取 PDF 文件信息 ()
    Dim PD As Acrobat.AcroPDDoc, result As Boolean
    Set PD = GetObject("", "AcroExch.PDDoc")
    With PD
        .Open "C:\temp\三国演义 .pdf"
        Debug.Print " 文件名: ", .GetFileName
        Debug.Print " 总页数 ", .GetNumPages
        Debug.Print "Title: ", .GetInfo("Title")
        Debug.Print "Creator: ", .GetInfo("Creator")
        Debug.Print "Keywords: ", .GetInfo("Keywords")
        Debug.Print "Subject: ", .GetInfo("Subject")
        Debug.Print "Author: ", .GetInfo("Author")
        Debug.Print "Created: ", .GetInfo("Created")
        Debug.Print "Modified: ", .GetInfo("Modified")
        Debug.Print "Producer: ", .GetInfo("Producer")
        .Close
    End With
End Sub
```

运行上述过程，立即窗口的结果如图 11-19 所示。

Acrobat 软件打开该 PDF 文件，单击菜单【文件 / 属性】，可以看到该文档的 PDF 属性与 VBA 执行的结果是一致的，如图 11-20 所示。

```
立即窗口
文件名:        三国演义.pdf
总页数          10
Title:         刘永富用VBA生成的小说
Creator:       Microsoft?Word 2013
Keywords:      三国 VBA
Subject:       三国演义前10页
Author:        刘永富
Created:
Modified:
Producer:      Microsoft?Word 2013
```

图 11-19　获取 PDF 文档信息



图 11-20 手工查看 PDF 文档属性

下面的过程自动在后台设置 PDF 文档的部分属性。

```
Sub 后台设置 PDF 文件属性 ()
    Dim PD As Acrobat.AcroPDDoc, result As Boolean
    Set PD = GetObject("", "AcroExch.PDDoc")
    With PD
        .Open "C:\temp\ 三国演义 .pdf"
        .SetInfo "Keywords", " 水浒传  西游记 "
        .SetInfo "Author", " 刘  行 "
        .Save nType:=Acrobat.PDSaveFlags.PDSaveFull, sFullPath:="C:\temp\ 三国演
义 AddProperties.pdf"
        .Close
    End With
End Sub
```

代码分析：上述代码更改了关键词和作者这两个属性，需要注意的是，AcroPDDoc 的 Save 方法中只能另存到另一个 PDF 文档中，不能把修改保存到本文档。

11.8.2 裁剪页面

PDF 文档的页面可以裁剪，所谓裁剪页面，就是在原先页面中画一个矩形框，保留矩形框围起来的部分，矩形外侧四周变得不可见。

在 Acrobat 软件中依次选择 Adobe Acrobat 的菜单【文档 / 裁剪页面】，弹出“裁剪页面”对话框。

从“裁剪页面”对话框可以看出，一般的 PDF 页面的宽度约 600 点（Points），高度约 800 点，如图 11-21 所示。

使用代码可以自动裁剪页面。AcroPDDoc 对象的 CropPages 方法可以同时裁剪一个文

档中的多张页面，AcroPDPage 对象的 CropPage 方法可以裁剪指定的一页。

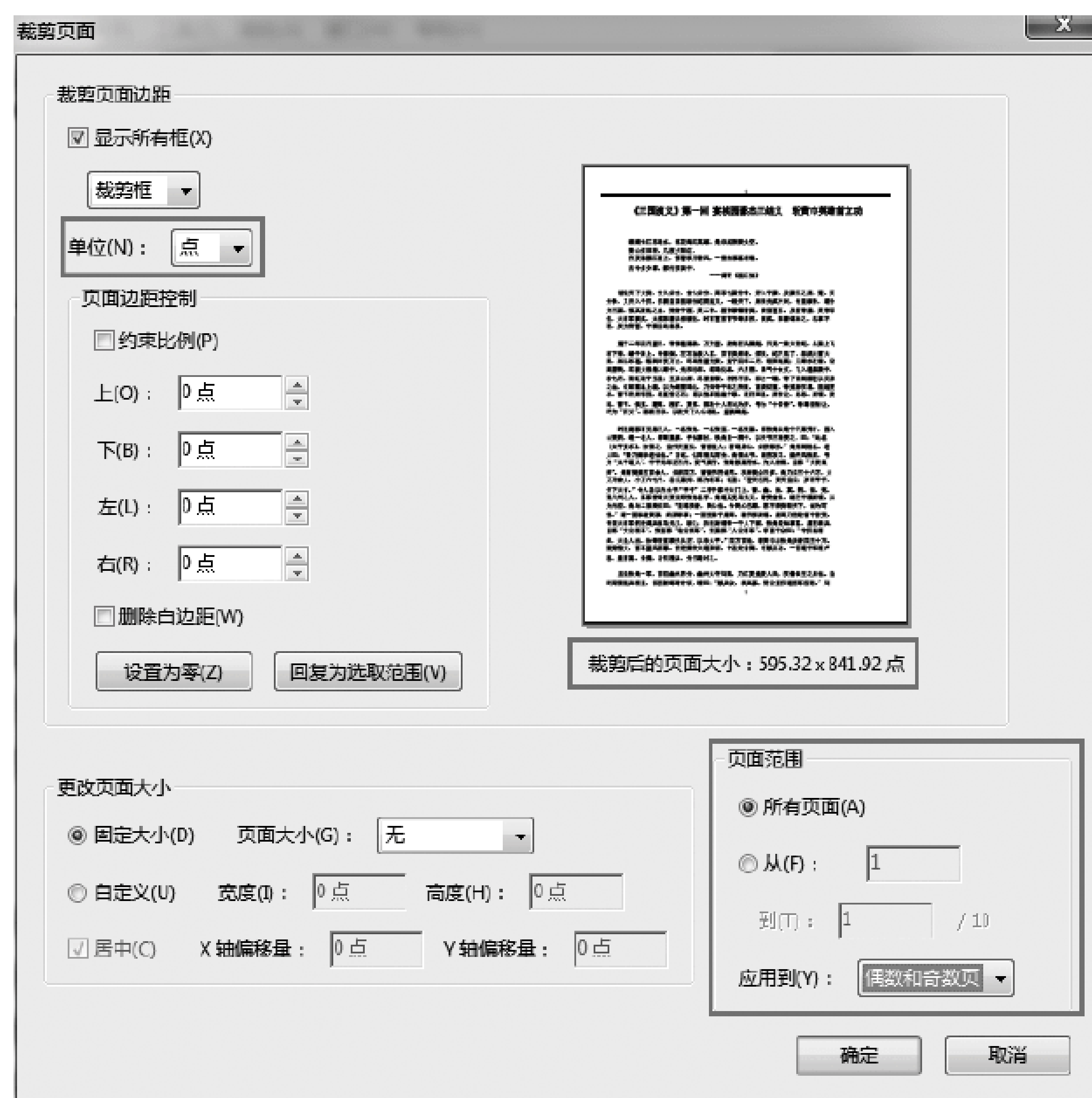


图 11-21　“裁剪页面”对话框

CropPages 方法的语法格式如下。

```
Function CropPages(nStartPage As Long, nEndPage As Long, nOddOrEvenPagesOnly
As Integer, iAcroRect As Object) As Boolean
```

参数说明如下。

- nStartPage 为开始页，如果从首页开始裁剪，设为 0。
- nEndPage 为结束页，如果裁剪到最后一页，设为 AcroPDDoc 对象的 GetNumPages。
- nOddOrEvenPagesOnly，可以设置为 0（开始页到结束页之间的所有页面）、1（奇数页）、2（偶数页）。
- iAcroRect 是一个 AcroRect 矩形对象，该对象需要在裁剪前创建，并设置其 Left、Right、Bottom、Top 属性，用来设定裁剪区域。

如果裁剪成功，则该方法返回 True。

例如，Doc.CropPages(0, 9, 2, rect) 表示把第 0、2、4、6、8 页按照 rect 的规定进行裁剪。

下面讲述一下 AcroRect 对象的设定方法，PDF 文档的某一页面放在一个直角坐标系中，其中点 C 的坐标为（600,800）。

假设裁剪区域是由矩形 ABCD 构成，点 A 的坐标为（200,300）。那么裁剪矩形对象 rect 的 Left 和 Right 是点 A 和点 B 的横坐标，Bottom 和 Top 是点 A 和点 D 的纵坐标。按照此矩形裁剪后，该页面只能看到矩形围起来的区域（深色所示区域），如图 11-22 所示。

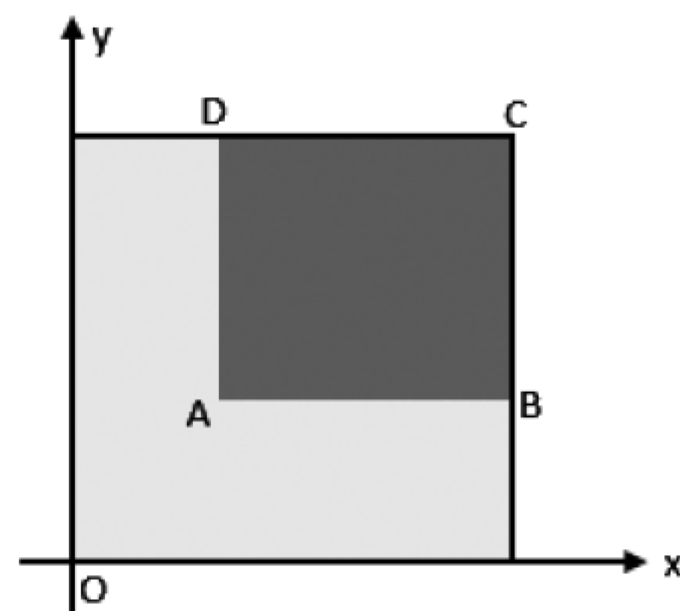


图 11-22　裁剪区域示意图

下面的程序把“三国演义 .pdf”文档的偶数页进行裁剪。

```
Sub 裁剪页面 ()
    Dim SourceFile As String
    Dim PD As Acrobat.AcroPDDoc, rect As Acrobat.AcroRect, result As Boolean
    Dim i As Integer
    Set PD = GetObject("", "AcroExch.PDDoc")
    Set rect = New Acrobat.AcroRect
    SourceFile = "C:\temp\ 三国演义 .pdf"
    PD.Open szFullPath:=SourceFile
    With rect
        .Left = 200 'Left 和 Right 是点 A 和点 B 的横坐标
        .Right = 600
        .bottom = 300 'Bottom 和 Top 是点 A 和点 D 的纵坐标
        .Top = 800
    End With
    result = PD.CropPages(0, 9, 2, rect)
    If result Then
        PD.Save nType:=Acrobat.PDSaveFlags.PDSaveFull, sFullPath:="C:\temp\Trim. df"
    End If
    PD.Close
End Sub
```

运行上述程序，在文件夹中生成一个 Trim.pdf 文件，在 Acrobat 中打开后，可以看到从首页起，每隔一页被裁剪，裁剪后的页面显然变小，如图 11-23 所示。

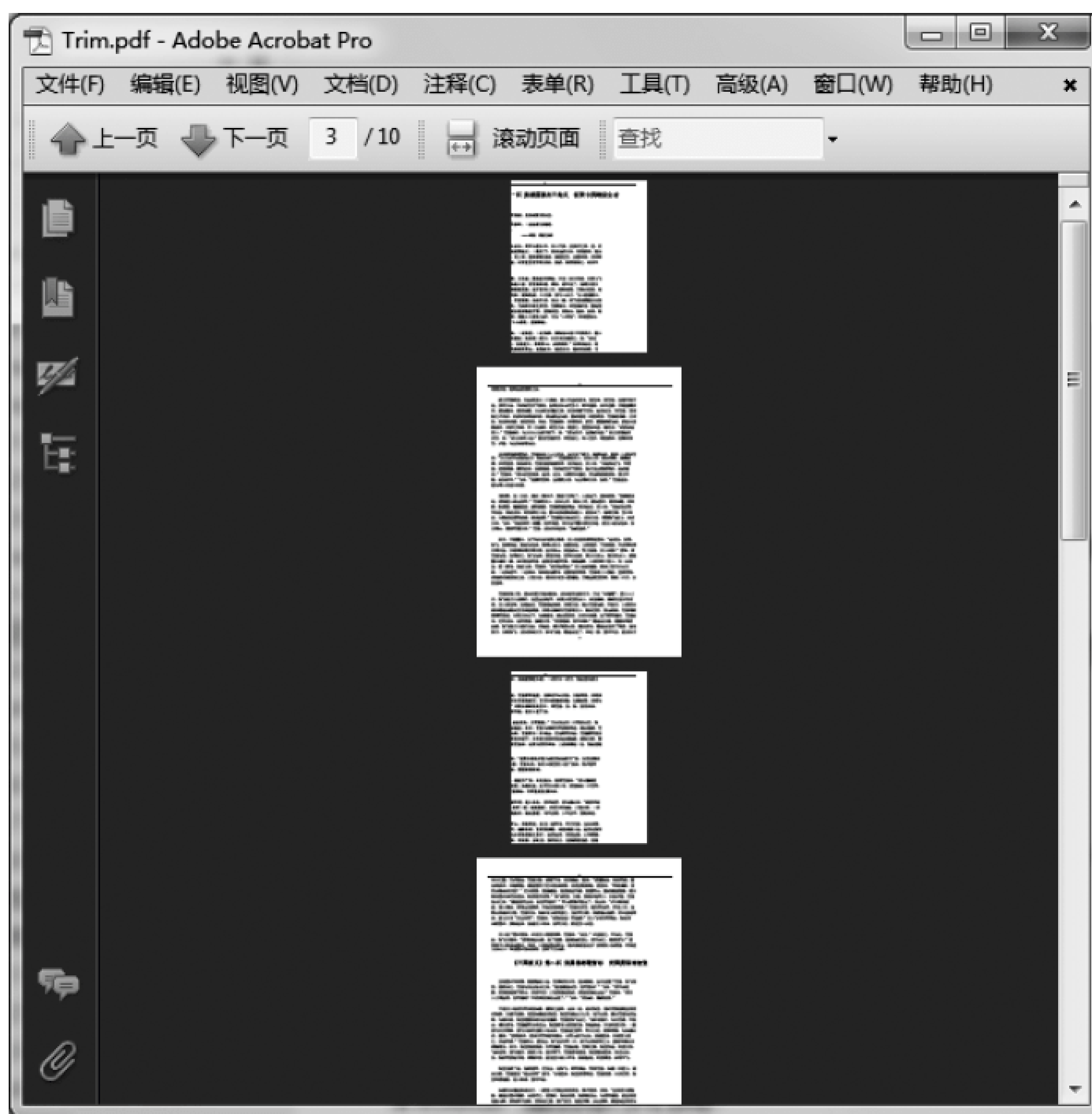


图 11-23 批量裁剪偶数页

需要注意的是，页面被裁剪后，裁剪掉的边缘部分仍然在文档中，只是看不见了而已，如果要把裁剪了的页面还原为初始状态，只需要把 AcroRect 对象的 Left 和 Bottom 都设置为 0，Right 和 Top 都设置为页面初始宽度和初始高度，再执行一次 CropPages 方法即可。

11.8.3　删除页面

AcroPDDoc 对象的 DeletePages 方法可以删除 PDF 文档中指定页码范围的页面。

```
Sub 删除页面()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroAVDoc
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open "C:\temp\三国演义.pdf"
    Debug.Print PD.GetFileName
    result = PD.DeletePages(nStartPage:=3, nEndPage:=6)
    Set Doc = PD.OpenAVDoc("")
    Doc.Close bNoSave:=False
End Sub
```

上述过程执行后，原来 10 页的 PDF 文档，删除其中第 3 ～ 6 页，也就是删除实际的第 4 ～ 7 页。

PD.OpenAVDoc("") 表示在 Acrobat 中显示该文档，最后保存。

11.8.4　移动页面

AcroPDDoc 对象的 MovePage 方法可以调整 PDF 文档中指定页面的位置。

```
Sub 移动页面()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroAVDoc
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open "C:\temp\三国演义.pdf"
    result = PD.MovePage(lMoveAfterThisPage:=5, lPageToMove:=2)
    Set Doc = PD.OpenAVDoc("")
    Doc.Close bNoSave:=False
End Sub
```

以上代码把文档中原来的第 2 页移动为第 5 页，也就是实际文档的第 3 页移动到第 6 页的位置。

有些情况下，需要把现有 PDF 文档中各个页面倒序排列。运行下面的代码可以把一个 PDF 文档各个页面倒序重新排列。

```
Sub 倒序重排()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroAVDoc, result As Boolean
    Dim count As Integer, i As Integer
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open "C:\temp\三国演义.pdf"
    count = PD.GetNumPages
    Set Doc = PD.OpenAVDoc("")
    For i = count - 2 To 0 Step -1
        result = PD.MovePage(lMoveAfterThisPage:=count - 1, lPageToMove:=i)
    Next i
End Sub
```

代码分析：由于 MovePage 方法的参数 lMoveAfterThisPage 表示放置于哪一页之后，因此需要从倒数第 2 页开始依次放到文档尾部。

11.8.5 插入页面

AcroPDDoc 对象的 InsertPages 方法可以把其他 PDF 文档中的若干页面插入现有 PDF 文档中。该方法的语法如下。

```
Function InsertPages(nInsertPageAfter As Long, iPDDocSource As Object,
lStartPage As Long, lNumPages As Long, lInsertFlags As Long) As Boolean
```

例如 result = FileA.InsertPages(nInsertPageAfter:=3, iPDDocSource:=FileB, lStartPage:=0, lNumPages:=2, lInsertFlags:=0) 的含义是在文件 FileA 的第 3 页后面插入文件 FileB 的第 0 页之后的连续 2 页。

如果插入页面成功，则返回布尔值 True。

下面的代码依次打开“三国演义 .pdf”和“西游记 .pdf”，然后把“西游记 .pdf”文档的所有页面插入“三国演义 .pdf”的第 3 页以后。

```
Sub 插入页面 ()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroAVDoc, result As Boolean
    Dim xyj As Acrobat.AcroPDDoc
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open "C:\temp\ 三国演义 .pdf"
    Set xyj = GetObject("", "AcroExch.PDDoc")
    xyj.Open "C:\temp\ 西游记 .pdf"
     result = PD.InsertPages(nInsertPageAfter:=3, iPDDocSource:=xyj, lStartPage:=
0, lNumPages:=xyj.GetNumPages, lInsertFlags:=True)
    Set Doc = PD.OpenAVDoc("")
    Doc.Close bNoSave:=False
End Sub
```

也可以把一个 PDF 文档中的一部分页面分离出去形成新文档。

下面的程序创建一个空白文档，然后把“三国演义 .pdf”文档的前 3 页插入新文档中。

```
Sub 分离部分页面 ()
    Dim PD As Acrobat.AcroPDDoc, NewDoc As Acrobat.AcroPDDoc, result As Boolean
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open "C:\temp\ 三国演义 .pdf"
    Set NewDoc = New Acrobat.AcroPDDoc
    NewDoc.Create ' 创建空白 PDF 文档
    result = NewDoc.InsertPages(nInsertPageAfter:=-1, iPDDocSource:=PD, lStartPage:=0,
lNumPages:=3, lInsertFlags:=0)
    If result Then
        NewDoc.Save nType:=Acrobat.PDSaveFlags.PDSaveFull, sFullPath:="C:\temp\
NewDoc.pdf"
        NewDoc.Close
    End If
    PD.Close
End Sub
```

运行上述程序，在磁盘下生成一个 NewDoc.pdf，该文档有 3 个页面。

因此，巧妙利用 InsertPages 方法，既可以把一个大文档拆分为多个小文档，也可以把

计算机中的多个 PDF 文档插入一个文档中，实现文档合并的目的。

11.8.6　拆分文档

假设“三国演义 .pdf”文档共有 10 页，现在要求每 2 页形成一个小文档，例如第 1 ~ 2 页形成“12.pdf”，第 3 ~ 4 页形成“34.pdf”，以此类推。

实现的原理是，首先打开源文档，然后在循环体中创建新文档、插入页面、关闭文档。

```
Sub 拆分文档 ()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroPDDoc, result As Boolean
    Dim i As Integer
    Set PD = GetObject("", "AcroExch.PDDoc")
    PD.Open szFullPath:="C:\temp\ 三国演义 .pdf"
    For i = 1 To PD.GetNumPages Step 2
        Set Doc = New AcroPDDoc
        Doc.Create
        result = Doc.InsertPages(nInsertPageAfter:=-1, iPDDocSource:=PD, lStartPage:=
i - 1, lNumPages:=2, lInsertFlags:=0)
        Doc.Save nType:=Acrobat.PDSaveFlags.PDSaveFull, sFullPath:="C:\temp\"
& i & (i + 1) & ".pdf"
        Doc.Close
    Next i
    PD.Close
End Sub
```

运行上述程序，在文件夹中生成 5 个以数字命名的小文档，每个文档包含 2 页，如图 11-24 所示。

名称	修改日期	类型	大小
12.pdf	2018/5/9 22:01	PDF 文件	488 KB
34.pdf	2018/5/9 22:01	PDF 文件	487 KB
56.pdf	2018/5/9 22:01	PDF 文件	349 KB
78.pdf	2018/5/9 22:01	PDF 文件	487 KB
910.pdf	2018/5/9 22:01	PDF 文件	349 KB

图 11-24　自动拆分文档

11.8.7　合并文档

合并文档是把计算机中已存在的多个 PDF 文档的所有页面插入一个文档中。

下面的程序首先创建一个空白文档，然后在循环结构中依次打开每个 PDF 文档，每打开一个，就把全部内容插入空白文档中，最后保存、关闭所有文档。

```
Sub 合并文档 ()
    Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroPDDoc, result As Boolean
    Dim Files As Variant
    Dim i As Integer
    Set Doc = New AcroPDDoc
    Doc.Create
    Files = Array("C:\Private\1.pdf", "C:\Private\5.pdf", "C:\Private\8.pdf")
```

```
        For i = 0 To UBound(Files)
            Set PD = GetObject("", "AcroExch.PDDoc")
            PD.Open szFullPath:=Files(i)
            result = Doc.InsertPages(nInsertPageAfter:=Doc.GetNumPages - 1, iPDDocSource:=
PD, lStartPage:=0, lNumPages:=PD.GetNumPages, lInsertFlags:=0)
            PD.Close
        Next i
        Doc.Save nType:=Acrobat.PDSaveFlags.PDSaveFull, sFullPath:="C:\Private\Doc.pdf"
        Doc.Close
    End Sub
```

运行上述程序，在磁盘下生成一个 DOC.pdf 文件，该文件中的内容是原先 3 个 PDF 文档合并的结果。

11.8.8 替换页面

AcroPDDoc 对象的 ReplacePages 方法与 InsertPages 非常类似，可以把其他的 PDF 文档中的若干页面插入现有 PDF 文档中。与 InsertPages 方法的区别是，插入的新页面会代替旧的页面。

下面的代码把“西游记 .pdf”文档中从第 2 页开始连续的 3 页替换“三国演义 .pdf”文档中第 4 页开始的连续 3 页。

```
    Sub 替换页面()
        Dim PD As Acrobat.AcroPDDoc, Doc As Acrobat.AcroAVDoc, result As Boolean
        Dim xyj As Acrobat.AcroPDDoc
        Set PD = GetObject("", "AcroExch.PDDoc")
        PD.Open "C:\temp\三国演义.pdf"
        Set xyj = GetObject("", "AcroExch.PDDoc")
        xyj.Open "C:\temp\西游记.pdf"
        result = PD.ReplacePages(nStartPage:=4, iPDDocSource:=xyj, lStartSourcePage:=
2, lNumPages:=3, bMergeTextAnnotations:=False)
        Set Doc = PD.OpenAVDoc("")
        Doc.Close bNoSave:=False
    End Sub
```

运行上述过程后，“三国演义 .pdf”文档的第 4、5、6 页连续 3 页变成“西游记 .pdf”文档中的 3 页。

AcroPDDoc 的 AcquirePage 属性还可以返回一个 AcroPDPage 对象。

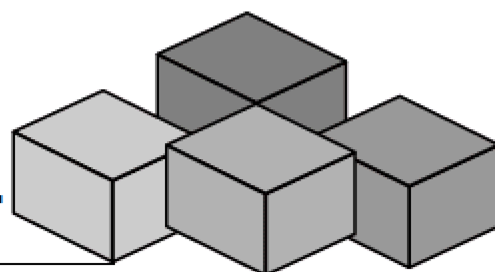
以上内容的源代码文件为“实例文档 72.xlsm”。

11.9 本章小结

Acrobat 是一个用来查看和编辑 PDF 文档的专业软件，安装了该软件后在计算机中产生 Acrobat 对象库，便于在其他编程语言中调用。

使用 AcroApp 应用程序对象和 AcroAVDOC 文档对象，可以自动打开、关闭 PDF 文档，也可以获取、设置 PDF 文档的有关属性。

第 12 章 自动发送邮件



邮件的收发是现在日常办公不可缺少的一部分工作，特别是在大型企业中，人与人之间的邮件往来非常频繁。再加上邮件的到达和发送具有不确定性的特点，因此人工每天应对这些邮件是非常烦琐的。

如果在办公计算机上安装了 Outlook，配置好账户后，就可以利用 Outlook VBA 自动发送邮件，也可以自动处理接收到的邮件。

但是考虑到很多没有用过 Outlook，也不太知道如何配置账户，因此本章介绍用 CDOMessage 这个外部引用，从而达到用 VBA 自动发送邮件的目的。

如果调用 Outlook 发送邮件，那么代码所在计算机必须事先配置好 Outlook 账户才行，但使用 CDOMessage，计算机有代码就够了，也就是说，邮箱的配置过程在 VBA 代码中完成。

然而，无论是哪一种方式代发邮件，必须通过网页浏览器进入邮箱，手工进行相关设置，开启邮箱的 SMTP 服务才行，如果不开启 SMTP 服务，只能从网页浏览器登录邮箱，手工发信。

本章用到的外部引用和重要对象：

- Microsoft CDO for Windows 2000 Library
  - CDO.Configuration
  - CDO.Message

12.1 开启 POP3/SMTP 服务

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议，它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。SMTP 协议属于 TCP/IP 协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。通过 SMTP 协议所指定的服务器，就可以把 E-mail 寄到收信人的服务器上，整个过程只要几分钟。SMTP 服务器则是遵循

SMTP 协议的发送邮件服务器，用来发送或中转发出的电子邮件。

简单地说，POP 是用于接收邮件，SMTP 用来发送邮件。开启 SMTP 服务是为了获得用程序自动发信的许可，下面介绍几个常用邮箱的 SMTP 设置方法。

12.1.1 QQ 邮箱的 SMTP 设置

下面以 19488012@qq.com 这个账户为例，讲述一下如何开启 SMTP 服务。

首先从浏览器使用邮箱账户和密码登录邮箱，进入邮箱后单击【设置 / 账户】，如图 12-1 所示。



图 12-1 邮箱账户设置

向下滚动到【POP3/SMTP... 服务】，检查【POP3/SMTP 服务】是不是处于“已开启”状态，如果是关闭状态，则单击“开启”，如图 12-2 所示。



图 12-2 开启邮箱的 SMTP 服务

更改设置后，看一下网页页面顶端或底端是否有【保存】按钮，如果有，单击保存从而让更改生效。

对于 QQ 邮箱或者网易 163 邮箱，用代码代发邮件时，不使用邮箱的登录密码，而是使用授权码。因此，当开启 SMTP 服务后，一定要单击【生成授权码】，有的授权码是邮件服务器自动生成的一个字符串，有的授权码是由用户指定的。

12.1.2 查看邮箱服务器属性

自动发送邮件，必须知道 SMTP 服务器的相关属性设置，对于 QQ 邮箱，单击【如何使用 Foxmail 等软件收发邮件】，就可以从浏览器中打开相关的帮助信息。

如图 12-3 所示的是 Outlook 中配置邮箱的参数设定。

可以看到，SMTP 服务器的地址是：smtp.qq.com。



图 12-3　Outlook 邮箱配置界面

在【其他设置】中，可以看到使用 SMTP 发送邮件的端口号是 465（SSL 加密），如图 12-4 所示。

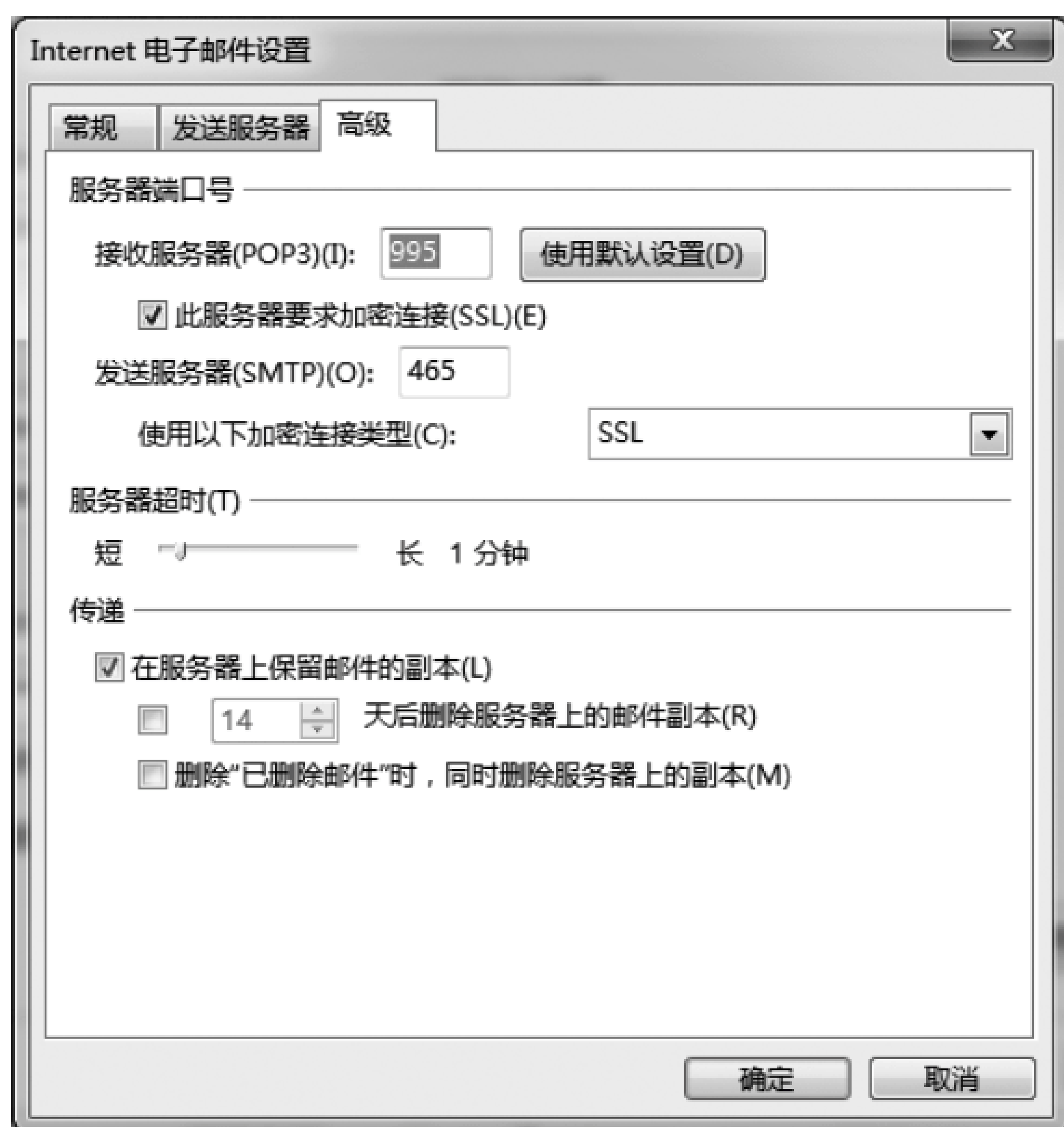


图 12-4　端口号设置界面

SSL（Secure Sockets Layer，安全套接层）及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。

是否要求 SSL 加密，所使用的端口号也有所不同，这一点要特别注意。

下面介绍其他常用邮箱的设置和属性查看方法。

12.1.3 网易 163 邮箱的 SMTP 设置

网易 163 邮箱也采用授权码代替登录密码，端口号是 25，如图 12-5 所示。



图 12-5 在 163 邮箱中获取授权码

12.1.4 日本雅虎邮箱的 SMTP 设置

日本雅虎邮箱的 SMTP 服务器为 smtp.mail.yahoo.co.jp，SSL 加密端口号为 465，如图 12-6 所示。

フィルターと受信通知
POP/IMAPアクセス…
自動返信
便利機能と広告最適化…
アカウント管理
メールアカウント
メールアドレス変更
セーフティーアドレス
追加メールアドレス
マイネームアドレス
セキュリティー強化
迷惑メール対策
受信拒否
なりすましメール拒否
セキュリティーパック
ウイルスチェック
海外からのアクセス制限

メールソフトでサーバー設定を行うには、次の情報が必要です。
今後の参照用として、このページを印刷し、保管してください。
戻る

IMAPアクセス（モバイル端末限定）

・iPhone/iPad/iPod touchの標準メーラーをお使いの方
iPhone/iPad/iPod touchでの設定方法
・それ以外のメーラーをお使いの方
Android端末などでIMAPアクセスを利用する

サーバー設定	
受信メール（IMAP）サーバー	imap.mail.yahoo.co.jp
受信メール（IMAP）通信方法	SSL
受信メール（IMAP）ポート番号	993
送信メール（SMTP）サーバー	smtp.mail.yahoo.co.jp
送信メール（SMTP）認証方式	SMTP_AUTH
送信メール（SMTP）通信方法	SSL
送信メール（SMTP）ポート番号	465
アカウント名/ログイン名	ryueifu2009
メールアドレス	ryueifu2009@yahoo.co.jp
パスワード	Yahoo! JAPAN IDのパスワード

※メールソフト（Outlook、Becky!、Thunderbirdなど）でご利用の場合は、サポート対象外となります。

图 12-6 日本雅虎邮箱的 SMTP 设置

12.2 VBA 中使用 CDO

VBA 编程中，可以使用 CDO 来配置邮箱并发送邮件。

首先在 VBA 工程中添加引用“ Microsoft CDO for Windows 2000 Library”，如图 12-7 所示。

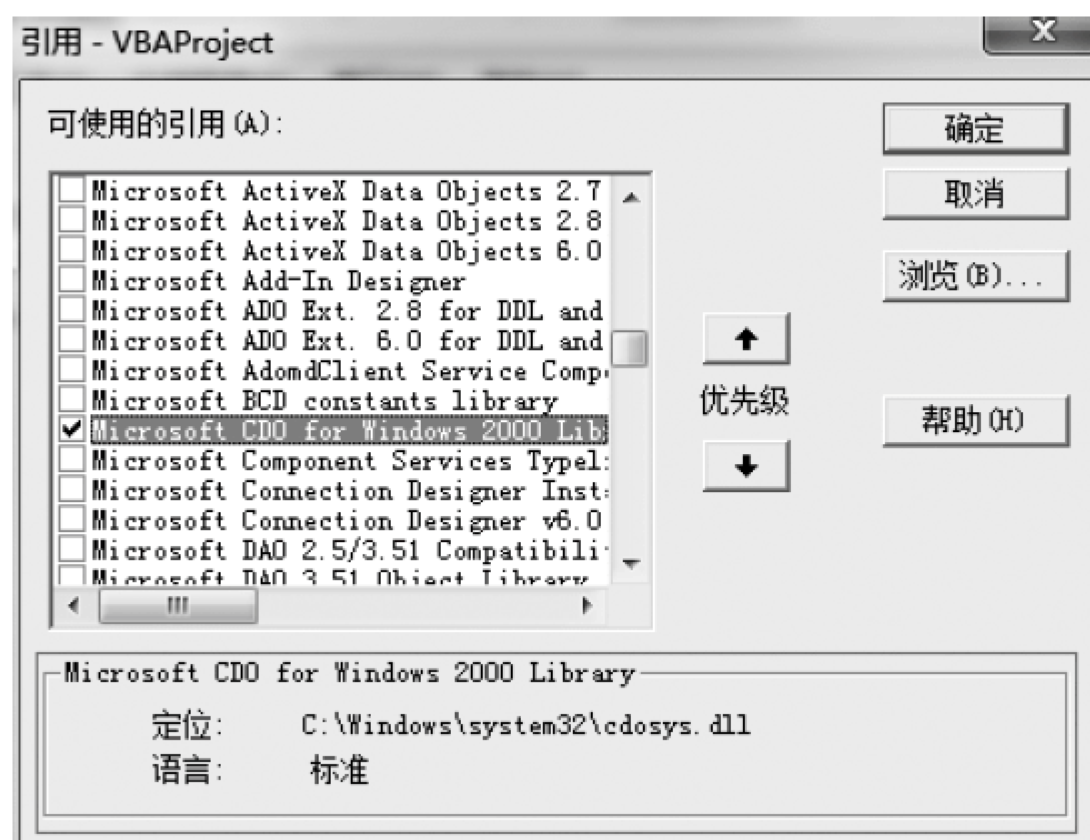


图 12-7 添加外部引用

添加 CDO 的引用后，就可以使用 Message 对象和 Configuration 对象了，其中 Message 对象用于设置一封邮件，而 Configuration 对象则用于配置发信账户。

12.2.1 配置发信账户

CDO 的 Configuration 对象有很多 Field（ADODB 中的字段），下面的过程用于配置 QQ 邮箱。

```
Public Mail As CDO.Message, Config As CDO.Configuration
Sub AccountConfig()
    Const nms As String = "http://schemas.microsoft.com/cdo/configuration/"
    Set Config = New CDO.Configuration
    With Config.Fields
        .Item(nms & "smtpusessl").Value = True                    '使用 SSL 加密
        .Item(nms & "sendusing").Value = 2                        '使用网络上的服务器
        .Item(nms & "smtpserver").Value = "smtp.qq.com"           'SMTP 服务器地址
        .Item(nms & "smtpserverport").Value = 25                  '端口号
        .Item(nms & "smtpauthenticate").Value = 1                 '服务器认证方式
        .Item(nms & "sendusername").Value = "19488012"            '发件人邮箱的用户名
        .Item(nms & "sendpassword").Value = "bmcigbnbsyitcbbi"'账户密码或授权码
        .Update                                                   '更新属性
    End With
End Sub
```

代码分析：需要注意的是，.Item(nms & "sendusername") = "19488012" 这一行代码只需要写上邮箱地址 @ 符号前面的部分即可。

运行上述过程，邮箱的配置信息就保存在公有变量 Config 中。

12.2.2 创建邮件

一封邮件的组成部分，主要包括如下几部分。

- 发件人的邮箱地址。
- 收件人地址、抄送地址、密送地址。
- 主题。
- 正文主体（纯文本或 HTML）。
- 附件。

CDO 中使用 Message 对象代表一封邮件，Message 对象除了设置上述邮件的组成部分外，还需要关联到邮件的配置对象 Config。

下面的过程自动创建一封邮件并发送。

```
Sub CreateMail()
    Set Mail = New CDO.Message
    With Mail
        .Configuration = Config                    '与配置关联
        .From = "19488012@qq.com"                  '发件人邮箱
        .To = "32669315@qq.com"
        .CC = "32669315@qq.com"
        .BCC = "32669315@qq.com"
        .Subject = "使用 CDO 代发 QQ 邮箱"          '邮件主题
        '.TextBody = "上班通知：下周一统一放假！ "   '常规文本内容作为邮件正文
        .HTMLBody = "<h1>多彩邮件</h1><br/>" & "<a href='http://vba.mahoupao.net/
forum.php'>欢迎光临 VBA 马后炮论坛！ </a>"
        .AddAttachment ("C:\temp\siping.csv")     '增加附件
        .AddAttachment ("C:\temp\答题卡.rar")
        .Send
    End With
End Sub
```

代码分析：由于发信账户存储在公有变量 Config 中，因此，Mail.Configuration=Config 就把配置信息应用到该邮件中。

代码中，From、To、Subject、TextBody 或 HTMLBody 是必须设置的属性。其他要根据实际情况进行增删。

运行上述过程，在收件人的邮箱中，可以看到这封新邮件，邮件的正文中有一个一级标题，还有一个超链接，如图 12-8 所示。

以上讲述的两个过程，其中配置发件账户的过程只运行一次即可，而 CreateMail 过程则可以多次运行，从而发送多封邮件。

如果要使用其他发件人账户，适当修改 AccountConfig 过程即可。下面的过程配置新浪邮箱。

```
Sub AccountConfig()
    Const nms As String = "http://schemas.microsoft.com/cdo/configuration/"
    Set Config = New CDO.Configuration
```

```
    With Config.Fields
        .Item(nms & "smtpusessl").Value = True          ' 使用 SSL 加密
        .Item(nms & "sendusing") = 2                    ' 使用网络上的服务器
        .Item(nms & "smtpserver") = "smtp.sina.com"     'SMTP 服务器地址
        .Item(nms & "smtpserverport") = 25              ' 端口号
        .Item(nms & "smtpauthenticate") = 1             ' 服务器认证方式
        .Item(nms & "sendusername") = "ryueifu2018"     ' 发件人邮箱的用户名
        .Item(nms & "sendpassword") = "liuyongfu"       ' 账户密码或授权码
        .Update                                         ' 更新属性
    End With
End Sub
```



图 12-8　使用 CDO 发来的多彩邮件

服务器地址、端口号、发件人用户名、密码或授权码需要修改。

然后基于上面创建好的 Config 再发一封信。

```
Sub CreateMail()
    Set Mail = New CDO.Message
    With Mail
        .Configuration = Config                  ' 与配置关联
        .From = "ryueifu2018@sina.com"           ' 发件人邮箱
        .To = "32669315@qq.com"
        .Subject = " 使用 CDO 代发新浪邮箱 "       ' 邮件主题
        .TextBody = " 上班通知：下周一统一放假！ "  ' 常规文本内容作为邮件正文
        .Send
    End With
End Sub
```

运行上述过程，收件人邮箱中多了一封邮件，该邮件的正文是纯文本，因为代码中用的是 TextBody，如图 12-9 所示。



图 12-9 使用 CDO 代发新浪邮件

以上内容的源代码文件为“实例文档 81.xlsm”。

12.2.3 错误处理

如果邮箱的配置信息有问题，或者邮件的各个组成部分不合适，就会导致 Mail 的 Send 方法失败，如图 12-10 所示。



图 12-10 发信失败

出错后，结束程序运行，根据错误信息分析原因并修改端口号等属性重新尝试。

12.2.4 窗体版的邮件客户端

在实际使用中，可以把配置邮件的过程、发送邮件的过程改编成带参数的形式，从而方便调用。例如下面这句。

```
Sub MySinaMail(username As String, password As String)
```

也可以通过用户窗体和控件，制作个性化的邮件发送系统。笔者用 VB6 制作的 SendMail 就是调用 CDO 对象实现的，如图 12-11 所示。

可以从本书配套资源中下载“SendMail.rar”。



图 12-11　窗体版的邮件客户端

12.3　其他语言调用 CDO

CDO 作为 Windows 系统的一个公共对象库，除了能在 VBA、VB6 中引用以外，还可以用在 .Net 语言中。

只需要把 VBA 版的代码的语法改编成其他语言的语法即可。

12.3.1　VB.Net 调用 CDO

下面的项目在 Visual Studio 中创建了一个窗体应用程序，窗体上放置一个按钮。然后为项目添加“Microsoft CDO for Windows 2000 Library”的外部引用。

```
Imports CDO
Public Class Form1
    Private Config As CDO.Configuration
    Private Mail As CDO.Message
    Sub AccountConfig()
        Const nms As String = "http://schemas.microsoft.com/cdo/configuration/"
        Config = New CDO.Configuration
        With Config.Fields
            .Item(nms & "smtpusessl").Value = True              ' 使用 SSL 加密
            .Item(nms & "sendusing").Value = 2                  ' 使用网络上的服务器
            .Item(nms & "smtpserver").Value = "smtp.qq.com"     'SMTP 服务器地址
            .Item(nms & "smtpserverport").Value = 25            ' 端口号
            .Item(nms & "smtpauthenticate").Value = 1           ' 服务器认证方式
            .Item(nms & "sendusername").Value = "19488012"      ' 发件人邮箱的用户名
            .Item(nms & "sendpassword").Value = "bmcigbnbsyitcbbi" ' 账户密码或授权码
            .Update()                                           ' 更新属性
        End With
    End Sub
    Sub CreateMail()
        Mail = New CDO.Message
        With Mail
            .Configuration = Config                             ' 与配置关联
```

```
                .From = "19488012@qq.com"                    ' 发件人邮箱
                .To = "32669315@qq.com"
                .CC = "32669315@qq.com"
                .BCC = "32669315@qq.com"
                .Subject = " 使用 VB.Net 代发 QQ 邮箱 "          ' 邮件主题
                '.TextBody = " 上班通知：下周一统一放假！ "      ' 常规文本内容作为邮件正文
                .HTMLBody = "<h1> 多彩邮件 </h1><br/>" & "<a href='http://vba.mahoupao.
net/forum.php'> 欢迎光临 VBA 马后炮论坛！ </a>"
                .AddAttachment("C:\temp\siping.csv")        ' 增加附件
                .AddAttachment("C:\temp\ 答题卡 .rar")
                .Send
            End With
        End Sub
        Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
            Call AccountConfig()
            Call CreateMail()
        End Sub
```

启动窗体后，单击按钮，邮件就自动发送出去了。

12.3.2 C# 调用 CDO

与 VB.Net 类似，在 Visual Studio 中创建一个窗体应用程序，放置一个按钮。为项目添加 CDO 这个外部引用，然后在模块顶部添加指令：using CDO;。

按钮的单击事件代码如下。

```
    private void button1_Click(object sender, EventArgs e)
            {
                CDO.Configuration Config;
                CDO.Message mail;
                const string nms = "http://schemas.microsoft.com/cdo/configuration/";
                string username = "19488012";
                string password = "bmcigbnbsyitcbbi";
                string smtp = "smtp.qq.com";
                int port    = 465;
                Config = new CDO.Configuration();
                Config.Fields[nms + "smtpusessl"].Value = 1;
                Config.Fields[nms + "sendusing"].Value = 2;
                Config.Fields[nms + "smtpserver"].Value = smtp;
                Config.Fields[nms + "smtpserverport"].Value = port;
                Config.Fields[nms + "smtpauthenticate"].Value = 1;
                Config.Fields[nms + "sendusername"].Value = username;
                Config.Fields[nms + "sendpassword"].Value = password;
                Config.Fields.Update();
                mail = new CDO.Message();
                mail.Configuration = Config;
                mail.From = "19488012@qq.com";
                mail.To = "32669315@qq.com";
                mail.Subject = "C# 代发 QQ 邮箱 ";
                mail.TextBody = " 仅仅是一个示例 ";
                mail.AddAttachment(@"C:\temp\ 答题卡 .rar");
                mail.Send();
            }
```

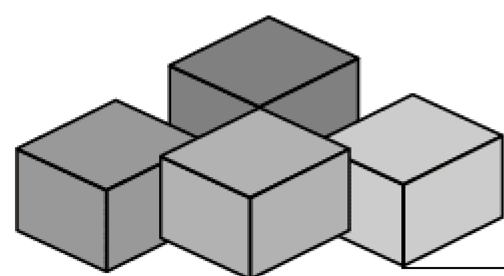
启动窗体，并单击按钮，邮件正常发出。

12.4　本章小结

邮箱只有开启了 POP/SMTP 服务，才能让其他程序调用。

不同类型的邮箱，SMTP 发送服务器、发送端口号有所不同，可以在网页上登录邮箱后查看到这些信息。

使用 CDO 发送邮件，需要把邮箱的账户信息写在程序代码中。



第 13 章 网页自动化

本章用到的外部引用和重要对象：

- ❑ Microsoft HTML Object Library
  - ➢ MSHTML.HTMLDocument
- ❑ Microsoft Internet Controls
  - ➢ SHDocVw.WebBrowser
  - ➢ SHDocVw.InternetExplorer
- ❑ Microsoft XML, v6.0
  - ➢ MSXML2.XMLHTTP60
- ❑ Microsoft WinHTTP Services, version 5.1
  - ➢ WinHttp.WinHttpRequest

13.1 网页自动化概述

Office VBA 编程，从字面上理解就是自动处理 Office 文档和本地文件的一种技术手段。但是由于 VBA 可以通过添加外部引用来扩展程序的功能，实现自动操作网页、获取和提交网络资源也成为可能。

当今时代是一个信息化时代，每时每刻都有大量的数据信息通过电子邮件、远程计算机和服务器、网页等媒介进行传输。这样必然导致办公人员的工作不仅仅是编辑、打印本地文档，还要每天从网页上查询、搜索有用的信息。光一个网页上就有成千上万个元素，每天用眼睛和双手浏览众多的网页势必给人们带来巨大的压力。

因此，实现网页的自动化操作，解放办公人员的双手，节省人力和时间成本的需求日益受到关注，网络数据抓取和分析也成为诸多编程语言的热点。

13.1.1 网页自动化包含的内容

网页自动化的目的和意义，就是解决人们上网的需求，把人和浏览器的交互过程转换为

程序代码。能够实现的自动化内容如下。

- 自动打开指定网址的网页。
- 自动分析网页内容。
- 自动下载、保存网页资源。
- 自动填写、提交表单。
- 自动提交数据、自动登录论坛。

凡是和网页、服务器有关的操作，都可以考虑使用编程的方式代替手工操作。

网页自动化与桌面程序自动化有很大的不同，桌面程序自动化主要靠 API 函数向指定句柄的窗口或控件发送鼠标单击或按键消息，自动化程序的宿主和操控的对象处于不同的进程。虽然网页自动化也需要鼠标和键盘操作，但主要是通过获取到网页元素，并且直接赋值或调用网页元素的方法实现自动化。

13.1.2　网页自动化开发所需知识和技能

网页是一种相对“特殊”并且“复杂”的对象，可以被看成由多种类型的元素构成的文档树，其后台语言是 HTML，但是表现给用户的页面是一个五彩缤纷的多媒体世界。

虽然可以从字符串的角度去思考网页内容，但是一般情况下每个网页的源代码相当长，使用一般的字符串处理技术分析网页相当烦琐，而且不准确。

因此，顺利开展网页自动化编程，需要的知识和技能如下。

- HTML 基础。
- HTML DOM 对象模型。
- 字符串处理函数和正则表达式。
- 会用浏览器开发工具查找和分析网页元素。
- 会用浏览器开发工具获取网页请求信息。

网页的后台是一个以 <html> 开始、</html> 结束、有一定组织结构的树状文档，从 HTML 的角度，< 和 > 括起来的称为“标签”。然而，从 HTML DOM 对象的角度，称为“元素”更为贴切。

HTML DOM 是用来描述一个网页文档组织结构的对象，通过使用 HTML DOM 可以快速定位到网页中的一个或者一组元素。

13.1.3　VBA 开发网页自动化的优势

微软公司提供了浏览器对象（WebBrowser、Internet Explorer）接口和发送 HTTP 请求的对象（XMLHTTP、WinHttp 等）接口。这些接口可以被微软公司的很多种编程语言调用，例如 Office VBA、VB6、.Net 语言。

虽然其他编程语言也能实现网页自动化和网页抓取，但门槛相对较高，开发出的自动化产品部署复杂。

使用 VBA 进行网页自动化的开发，只需要 VBA 开发环境和浏览器就够了，因此具备开发成本低、兼容性好、产品的分发部署简单等很多优点。

13.1.4 本章主要内容

本章总体上分为三大部分：认识 HTML 文档、自动操作浏览器和网页、自动发送 HTTP 请求，如表 13-1 所示。

表 13-1 本章内容框架

<table>
<tr><td colspan="4">HTML 基础
网页的构成、元素之间的关系、元素的属性</td></tr>
<tr><td colspan="4">HTML DOM 对象模型
使用 GetElement 方面的方法获取元素、自动填写并提交表单</td></tr>
<tr><td colspan="2">浏览器的自动操作
使用浏览器的开发工具查找元素
自动输入内容、单击按钮和超链接</td><td colspan="2">HTTP 请求的发送和响应
使用浏览器的开发工具录制请求过程
请求头和请求体、响应头和响应消息</td></tr>
<tr><td>Internet Explorer
独立的网页</td><td>WebBrowser
内嵌的控件</td><td>XMLHTTP</td><td>WinHttp</td></tr>
<tr><td colspan="4">共同内容：延时等待、文件下载、url 的编码和解码</td></tr>
</table>

其中 Internet Explorer 和 WebBrowser 的 Document 属性可以直接转换为一个 HTML Document，而由 XMLHTTP、WinHttp 得到的 ResponseBody、ResponseText 作为字符串也可以产生一个 HTMLDocument。

其中，HTMLDocument 方面的重点和难点是网页元素的定位方法，识别网页元素的类型，调用网页元素的方法。

XMLHTTP 和 WinHttp 的应用方面主要内容包括网页源代码的获取，使用 SetRequestHeader 设置请求头、请求体的构造和发送、编码和解码、使用 getAllResponseHeaders 和 getResponseHeader() 获取响应头等内容。

13.2 HTML 基础

HTML 指的是超文本标记语言（Hyper Text Markup Language），使用标记标签来描述网页。

标记标签，也可以称为元素（Element）。与 XML 中的元素非常类似，HTML 中的元素表示如下。

```
< 元素名称 属性名称 1= 属性值 1> 文本 </ 元素名称 >
```

大体来看，元素的定义可以分为三部分。

- ❑ 元素的开始标签，以 <> 括起来，如果有其他属性，也要写在开始标签的内部。
- ❑ 元素的文本，夹在开始标签和结束标签之间。

- 元素的结束标签，以 </> 表示，结束标签中的元素名称必须与开始标签中的元素名称相同。

一个元素的代码也可以写作多行。

当一个元素不包含任何子元素、文本时，可以不写结束标签。形式如下。

```
<元素名称 属性名称1=属性值1/>
```

注意，后面的左尖括号前面有一个斜杠，表示这个元素的代码到此为止。

HTML 中，元素和元素之间的基本关系是父子关系与兄弟关系。

一个网页的主要元素框架形式如下。

```
<html>
    <head>
        <meta charset="utf-8"/>
        <title>网页标题</title>
    </head>
    <body>
    主体内容
    </body>
</html>
```

以上代码可以用记事本程序编辑并保存为扩展名为 .html 的网页文件。

一个 HTML 文档的根元素是 html，下面包含 head 和 body 两个子元素，其中 head 用来指定网页的编码、标题等信息，body 是网页的主体部分。

下面介绍 body 元素下面最常用的网页元素的写法。

13.2.1　标题

这里所说的标题类似于 Word 中的表示章节的大纲标题。HTML 中使用 <h1> ~ <h6> 表示 1 ~ 6 级标题。

以下代码定义了一个 1 级标题，这个元素的名称是 h1, 属性名称是 style，属性值是 color:red，该元素的内容是“刘白梦的简历”。

```
<h1 style="color:red">刘白梦的简历</h1>
```

13.2.2　注释

HTML 中注释的写法如下：

```
<!—注释内容 -->
```

被注释的部分不起任何作用，在网页中看不到注释的内容。

13.2.3　表格

HTML 中，使用 table 定义一个表格。

table 元素的子元素为 tr，每个 tr 元素表示表格的一行，tr 元素下面包含多个 td 元素。每个 td 元素表示该行的一个单元格。

```
<table border="1">
    <tr><td> 姓名 </td><td> 刘白梦 </td><td> 性别 </td><td> 女 </td></tr>
    <tr><td> 年龄 </td><td>25</td><td> 民族 </td><td> 汉族 </td></tr>
    <tr><td> 毕业院校 </td><td> 化工大学 </td><td> 专业 </td><td> 化学工程 </td></tr>
</table>
```

以上 HTML 代码中，border="1" 表示这个表格显示边框线，还可以看出该表格有 3 行，每行有 4 列。

以上代码显示在网页中的效果如图 13-1 所示。

姓名	刘白梦	性别	女
年龄	25	民族	汉族
毕业院校	化工大学	专业	化学工程

图 13-1　HTML 表格

因此，HTML 表格的三层递进元素依次为 table、tr 和 td。

如果要把表格的首行显示为标题行，有时候还用 th 代替 td。

```
<table border="1">
    <tr><th> 语言 </th><th> 程度 </th><th> 使用时长 </th></tr>
    <tr><td>Excel VBA</td><td> 熟练 </td><td>5 年 </td></tr>
    <tr><td>VB6</td><td> 熟练 </td><td>2 年 </td></tr>
    <tr><td>Python</td><td> 入门 </td><td>1 年 </td></tr>
</table>
```

以上 HTML 在网页中的显示效果如图 13-2 所示。

语言	程度	使用时长
Excel VBA	熟练	5年
VB6	熟练	2年
Python	入门	1年

图 13-2　使用标题行的表格

13.2.4　图像

HTML 使用 img 定义一个图像，img 元素最重要的属性是 src，用来指定所指向的图像文件地址。

以下代码表示在此处显示一幅图，图片的地址是 photo.jpg，这是一个相对路径。

```
<img src='photo.jpg'/>
```

13.2.5　超链接

HTML 中使用 a 定义一个超链接，href 属性指定单击该超链接后跳入的地方。a 元素的文本内容是显示在网页上的内容。

```
<a href="https://home.cnblogs.com/u/ryueifu-VBA/"> 个人主页 </a>
```

上述代码在网页中产生的效果如图 13-3 所示。

通常情况下，一个网页中的超链接非常多。这里产生一个问题，当用户单击了某个超链接，是在原有页面中打开超链接，还是在新窗口中打开超链接？这取决于 target 属性。

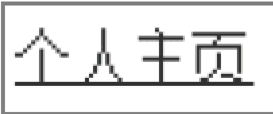


图 13-3　超链接

例如 target="_blank"，表示超链接被单击后，在新窗口中打开超链接的页面。

target="_self"，表示超链接被单击后，在原窗口中打开超链接的页面。

13.2.6　列表

HTML 的列表类似于 Word、PowerPoint 中的项目符号和列表。

无序列表用 ul 定义，使用 li 表示具体的条目。

```
<ul>
    <li>英语</li>
    <li>日语</li>
</ul>
```

以上代码定义了一个无序列表，显示效果如图 13-4 所示。

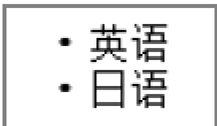


图 13-4　无序列表

有序列表用 ol 定义。

```
<ol>
    <li class="Chemical">北京化工厂－研发助理</li>
    <li class="Chemical">北京化学品公司－高级研发工程师</li>
    <li class="Programming">北京广告公司－编程开发岗位</li>
</ol>
```

以上代码的显示效果如图 13-5 所示。

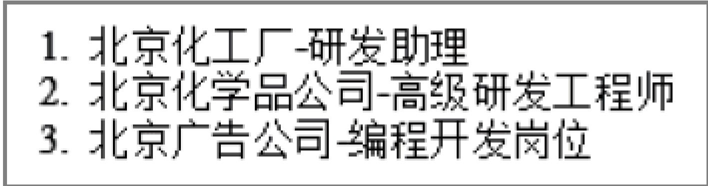


图 13-5　有序列表

13.2.7　表单控件

表单是允许用户在网页中进行交互的内容，例如在网页中提供文本框、按钮等。

HTML 中使用 form 定义一个表单，里面可以定义多个控件，表单控件统一使用 input 来定义。

下面的代码在表单中定义了 2 个文本框，用于接收用户输入的文本，最后定义了一个按钮。显示效果如图 13-6 所示。

图 13-6　表单

```
<form>
     职位名称：<input id="Keyword" name="InputField" type="text" value="VBA" />
    <br />
    期望薪资：<input id="Salary" name="InputField" type="text" value="8800" />
    <br />
     <button id="Search" name="Button1" onclick="alert('正在搜索，请稍候...')">搜索</button>
</form>
```

13.3 HTML DOM 对象模型

HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法。换言之，HTML DOM 是关于如何获取、修改、添加或删除 HTML 元素的标准。

在 VBA 中，向工程中添加“Microsoft HTML Object Library”的引用，就可以使用 HTML DOM 对象库中的对象类型、属性、方法等，如图 13-7 所示。

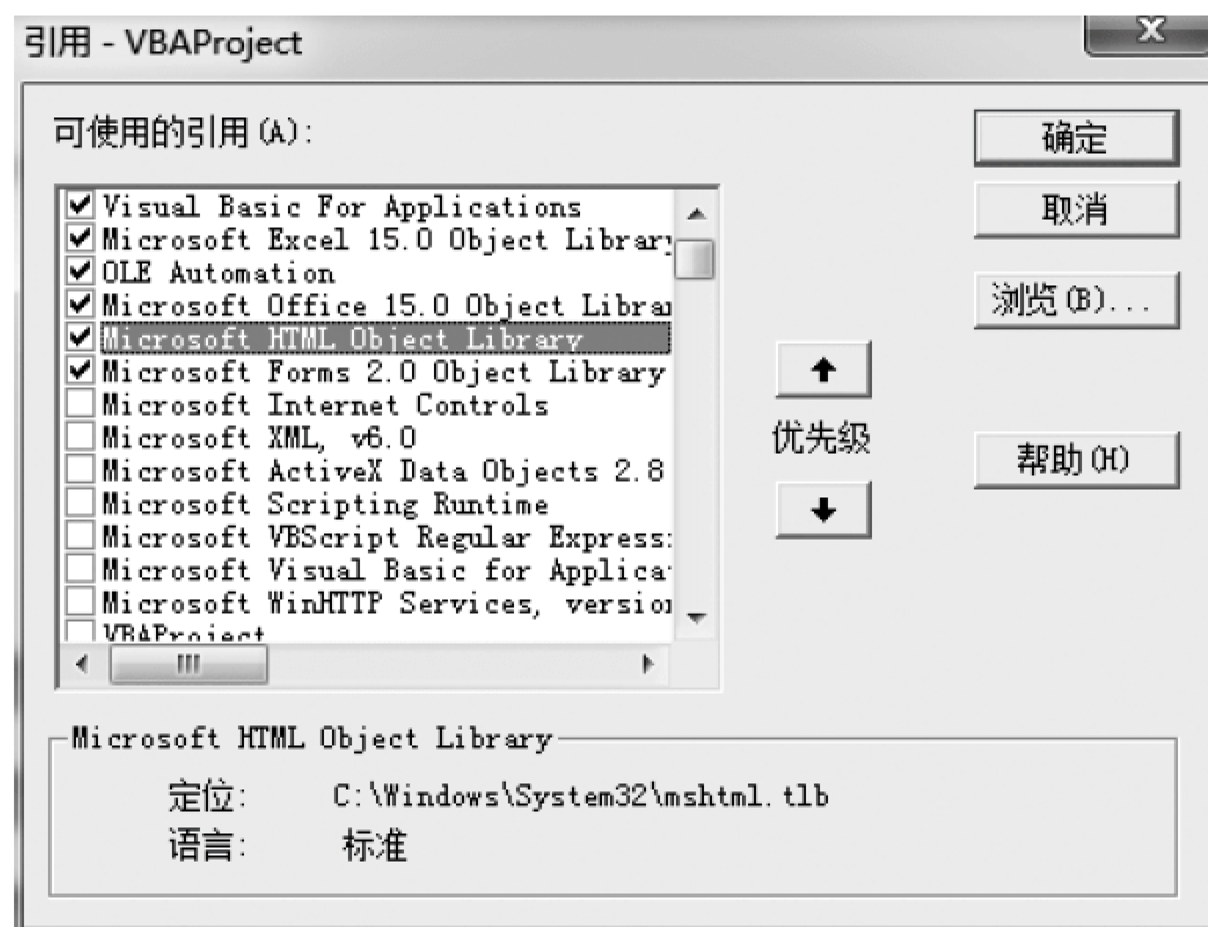


图 13-7 添加外部引用

一个网页的源代码可以被看作一个很长的字符串，如果用 VBA 的字符串处理函数，或者正则表达式来操作和访问 HTML 代码中的元素，则非常费事。

虽然 HTML 代码错综复杂，包含各式各样的元素，但还是有规律可循的，任何一个元素都不是孤立存在的，HTML 文档可以被看作一个由父子元素节点、兄弟元素节点连接而成的文档树。

HTML DOM 对象模型中，把一个网页的所有内容形成的文档定义为 HTMLDocument 对象，这个文档的根元素节点就是 html 节点。head 和 body 都是 html 节点的子节点。

13.3.1 使用 HTML DOM 创建网页

VBA 工程中加入了“Microsoft HTML Object Library”这个外部引用之后，在声明 HTML DOM 相关变量的时候，输入 MSHTML. 就可以看到所有对象类型名称。常用网页元素的对象类型，如表 13-2 所示。

表 13-2 常用网页元素的对象类型

节点名称	HTML DOM 对象类型	描述
<html>	MSHTML.HTMLHtmlElement	网页的根节点
<head>	MSHTML.HTMLHeadElement	网页的头部
<title>	MSHTML.HTMLTitleElement	网页的标题
<body>	MSHTML.HTMLBody	网页的主体

续表

节点名称	HTML DOM 对象类型	描　述
<a>	MSHTML.HTMLAnchorElement	超链接
 	MSHTML.HTMLBRElement	换行符
<button>	MSHTML.HTMLButtonElement	按钮
<caption>	MSHTML.HTMLTableCaption	表格的标题
<div>	MSHTML.HTMLDivElement	文档中的节
<form>	MSHTML.HTMLFormElement	表单
<frame>	MSHTML.HTMLFrameElement	frame 框架
<h1> 到 <h6>	MSHTML.HTMLHeaderElement	标题
<iframe>	MSHTML.HTMLIFrame	iframe 内联框架
<img>	MSHTML.HTMLImg	图像
<input>	MSHTML.HTMLInputElement	输入控件
	MSHTML.HTMLInputButtonElement	表单中的按钮
	MSHTML.HTMLInputTextElement	表单中的输入框
	MSHTML.HTMLInputFileElement	表单中的文件选择对话框
<label>	MSHTML.HTMLLabelElement	表单控件的标注
	MSHTML.HTMLLIElement	列表项目
	MSHTML.HTMLOListElement	有序列表
<option>	MSHTML.HTMLOptionElement	选择列表
<p>	MSHTML.HTMLParaElement	段落
<script>	MSHTML.HTMLScriptElement	客户端脚本
<select>	MSHTML.HTMLSelectElement	下拉列表
<table>	MSHTML.HTMLTable	表格
<tbody>	MSHTML.HTMLTableSection	表格主体内容
<td>、<th>	MSHTML.HTMLTableCell	表格的单元格
<tr>	MSHTML.HTMLTableRow	表格的行
	MSHTML.HTMLUListElement	无序列表
-	MSHTML.IHTMLElement	通用节点类型
-	MSHTML.IHTMLElementCollection	节点集合

一般情况下，编辑、创建网页是使用专业的网页编辑软件，手工书写而成。在 VBA 中，可以创建 HTMLDocument，并且逐级插入必要的元素，设置其相关属性，即可形成一个完整的网页。

使用 New 关键字创建一个新的 HTML 文档，此时该文档自动具备了基本框架（html、head、body 不需要创建节点）。因此，只需要在 Body 节点下面创建元素即可。

例如，要在 Body 中创建如下的超链接元素。

```
<A href="https://home.cnblogs.com/u/ryueifu-VBA/">刘白梦的个人主页</A>
```

对应的 VBA 代码如下。

```
    Dim a As MSHTML.HTMLAnchorElement
    Set a = HDoc.createElement("a")
    a.setAttribute strAttributeName:="href", AttributeValue:="https://home.cnblogs.
com/u/ryueifu-VBA/"
    a.innerText = "刘白梦的个人主页"
    body.appendChild a '把 a 附加到 Body 之下
```

上述代码的含义是，首先声明一个超链接元素，然后使用 HTML 文档创建一个 a 元素，设置 a 元素的 href 属性，并且设置 a 元素的文本内容，最后把 a 元素附加到 Body 中。

照着如上的逻辑和方法，就可以为 HTML 文档中加入各种类型的元素节点。

下面的程序自动创建一个 HTML 文档，并另存为本地网页文件。

```
    Sub 自动创建网页()
        Dim HDoc As MSHTML.HTMLDocument
        Set HDoc = New MSHTML.HTMLDocument
        Dim Body As MSHTML.HTMLBody
        Set Body = HDoc.Body

        Dim h1 As MSHTML.HTMLHeaderElement
        Set h1 = HDoc.createElement("h1")
        h1.innerText = "刘白梦的简历"
        Body.appendChild newChild:=h1                  '把 h1 元素附加到 Body 中

        Dim table As MSHTML.HTMLTable
        Dim tr As MSHTML.HTMLTableRow
        Dim td As MSHTML.HTMLTableCol
        Set table = HDoc.createElement("table")
        table.setAttribute strAttributeName:="border", AttributeValue:="1"
        Set tr = HDoc.createElement("tr")
        Set td = HDoc.createElement("td")
        td.innerText = "姓名"
        tr.appendChild td
        Set td = HDoc.createElement("td")
        td.innerText = "年龄"
        tr.appendChild td
        table.appendChild tr
        Set tr = HDoc.createElement("tr")
        Set td = HDoc.createElement("td")
        td.innerText = "刘白梦"
        tr.appendChild td
        Set td = HDoc.createElement("td")
        td.innerText = "25"
        tr.appendChild td
        table.appendChild tr
        Body.appendChild table                         '把 table 附加到 Body 中

        Dim img As MSHTML.HTMLImg
        Set img = HDoc.createElement("img")
        img.setAttribute strAttributeName:="src", AttributeValue:="Photo.jpg"
        img.setAttribute strAttributeName:="width", AttributeValue:=100
        img.setAttribute strAttributeName:="height", AttributeValue:=120
        Body.appendChild img                           '把 img 附加到 Body 中

        Dim p As MSHTML.HTMLParaElement
```

```
        Set p = HDoc.createElement("p")
        Body.appendChild p                          ' 插入一个空段落

        Dim a As MSHTML.HTMLAnchorElement
        Set a = HDoc.createElement("a")
        a.setAttribute strAttributeName:="href", AttributeValue:="https://home.cnblogs.
com/u/ryueifu-VBA/"
        a.innerText = "刘白梦的个人主页"
        Body.appendChild a                          ' 把 a 附加到 Body 中

        Dim form As MSHTML.HTMLFormElement
        Set form = HDoc.createElement("form")
        Dim text As MSHTML.HTMLInputTextElement
        Set text = HDoc.createElement("input")
        text.setAttribute strAttributeName:="id", AttributeValue:="Keyword"
        text.setAttribute strAttributeName:="type", AttributeValue:="text"
        text.setAttribute strAttributeName:="value", AttributeValue:="VBA"
        form.appendChild text
        Dim button As MSHTML.HTMLInputButtonElement
        Set button = HDoc.createElement("input")
        button.setAttribute strAttributeName:="id", AttributeValue:="Search"
        button.setAttribute strAttributeName:="type", AttributeValue:="button"
        button.setAttribute strAttributeName:="value", AttributeValue:="搜索"
        form.appendChild button
        Body.appendChild form                       ' 把 form 附加到 Body 中

        Open ThisWorkbook.Path & "\WebPage1.html" For Output As #1' 保存为网页文件
            Print #1, HDoc.DocumentElement.outerHTML
        Close #1
    End Sub
```

运行上述程序，会在工作簿所在路径生成 WebPage1.html，使用记事本查看内容，如图 13-8 所示。



```
<HTML>
<HEAD>
</HEAD>
<BODY>
    <P> </P>
    <H1>刘白梦的简历</H1>
    <TABLE border=1>
        <TR>
            <TD>姓名</TD>
            <TD>年龄</TD>
        </TR>
        <TR>
            <TD>刘白梦</TD>
            <TD>25</TD>
        </TR>
    </TABLE><IMG src="Photo.jpg" width=100 height=120>
    <P>
    </P>
    <A href="https://home.cnblogs.com/u/ryueifu-VBA/">刘白梦的个人主页</A>
    <FORM>
        <INPUT id=Keyword value=VBA>
        <INPUT id=Search type=button value=搜索>
    </FORM>
</BODY>
</HTML>
```

图 13-8　使用 VBA 自动创建网页

在 IE 浏览器中查看该网页，效果如图 13-9 所示。



图 13-9　在浏览器中打开网页

以上程序的源代码文件为“实例文档 82.xlsm”。

13.3.2　使用 HTML DOM 解析网页内容

无论一个网页的内容有多复杂，其根源都是 HTML 源代码，是一个很长的字符串。那么里面究竟包含了哪些元素，每个元素有哪些属性呢?

显然，用传统的字符串处理方式是不妥的。使用 HTML DOM 可以轻松获取 HTML 文档的方方面面。

位于计算机磁盘上的网页文件可以利用文本文件读取的方式读取网页代码，赋给 HTMLDocument 对象，而对于网络上的外部网页，在联网的情况下可以使用 XMLHttp 对象获取网页源代码。

在工作簿路径下有一个事先做好的网页“刘白梦的简历 .html”，下面的程序采用 XMLHttp 方式获取网页内容。

```
Sub 形成 HTML 文档 ()
    Dim X As MSXML2.XMLHTTP60, HDoc As MSHTML.HTMLDocument
    Set X = New MSXML2.XMLHTTP60
    With X
        .Open "GET", ThisWorkbook.Path & "\ 刘白梦的简历 .html", False
        .send
        Do Until .readyState = 4
            DoEvents
        Loop
        Set HDoc = New MSHTML.HTMLDocument
        HDoc.body.innerHTML = .responseText ' 把网页源代码赋给 HDoc，形成 DOM
        Debug.Print HDoc.DocumentElement.outerHTML
        ' 主要骨架元素
        Dim Root As MSHTML.HTMLHtmlElement '<html>...</html> 元素
        Dim Head As MSHTML.HTMLHeadElement '<head>...</head> 元素
        Dim Body As MSHTML.HTMLBody '<body>...</body> 元素
        Set Root = HDoc.DocumentElement
        Set Head = HDoc.Head
```

```
        Set Body = HDoc.Body
        Debug.Print Root.nodeName
        Debug.Print Head.nodeName
        Debug.Print Body.nodeName
    End With
End Sub
```

运行上述程序，立即窗口打印出如下网页源代码（为便于讲解，行首添加了行号）：

```
1.  <html>
2.      <head>
3.          <meta charset="utf-8"/>
4.          <title> 刘白梦的简历 </title>
5.      </head>
6.      <body>
7.          <!--1 级标题 -->
8.          <h1 style="color:red"> 刘白梦的简历 </h1>
9.          <!--2 级标题 -->
10.         <h2> 认真负责、实事求是！ </h2>
11.         <!-- 段落 -->
12.         <p> 基本信息 </p>
13.         <!-- 表格 -->
14.         <table border="1">
15.             <tr><td> 姓名 </td><td> 刘白梦 </td><td> 性别 </td><td> 女 </td></tr>
16.             <tr><td> 年龄 </td><td>25</td><td> 民族 </td><td> 汉族 </td></tr>
17.               <tr><td> 毕业院校 </td><td> 化工大学 </td><td> 专业 </td><td> 化学工程
                  </td></tr>
18.         </table>
19.         <!-- 图像 -->
20.         <p> 近照 </p>
21.         <img src='photo.jpg'/>
22.         <!-- 无序列表 -->
23.         <p> 外语水平 </p>
24.         <ul>
25.             <li> 英语 </li>
26.             <li> 日语 </li>
27.         </ul>
28.         <!-- 有序列表 -->
29.         <p> 工作经验 </p>
30.         <ol>
31.             <li class="Chemical"> 北京化工厂 - 研发助理 </li>
32.             <li class="Chemical"> 北京化学品公司 - 高级研发工程师 </li>
33.             <li class="Programming"> 北京广告公司 - 编程开发岗位 </li>
34.         </ol>
35.         <p> 计算机语言 </p>
36.         <table border="1">
37.             <tr><th> 语言 </th><th> 程度 </th><th> 使用时长 </th></tr>
38.             <tr><td>Excel VBA</td><td> 熟练 </td><td>5 年 </td></tr>
39.             <tr><td>VB6</td><td> 熟练 </td><td>2 年 </td></tr>
40.             <tr><td>Python</td><td> 入门 </td><td>1 年 </td></tr>
41.         </table>
42.        <br/>
43.         <!-- 超链接 -->
44.          <a href="https://home.cnblogs.com/u/ryueifu-VBA/" target="_blank">
             个人主页 </a>
45.         <br/>
```

```
46.         <!-- 表单 -->
47.         <p>职位搜索</p>
48.         <form>
49.             职位名称：<input id="Keyword" name="InputField" type="text"
                value="VBA" />
50.             <br />
51.             期望薪资：<input id="Salary" name="InputField" type="text"
                value="8800" />
52.             <br />
53.             <button id="Search" name="Button1" onclick="alert('正在搜索，请稍
                候...')">搜索</button>
54.         </form>
55.     </body>
56. </html>
```

接下来介绍一下 HTML 文档中的主要对象和管辖范围。

任何一个 HTML 文档有且只有一个 <html> 根节点，该节点下面包含 <head> 和 <body> 两个子节点，而网页中大多数的元素都是 <body> 节点的后代节点。

以上三个骨架对象的声明和赋值方法如下。

```
Dim Root As MSHTML.HTMLHtmlElement '<html>...</html>元素
Dim Head As MSHTML.HTMLHeadElement '<head>...</head>元素
Dim Body As MSHTML.HTMLBody '<body>...</body>元素
Set Root = HDoc.DocumentElement
Set Head = HDoc.Head
Set Body = HDoc.Body
```

其中，HDoc 是一个 HTML 文档对象。

如果以上面的网页源代码为例，那么 Root 对象的范围是第 1 ~ 56 行，Head 对象的范围是第 2 ~ 5 行，Body 对象的范围是第 6 ~ 55 行。

打印它们的 OuterHTML 属性，就可以了解各自的范围。

13.3.3 获取和定位网页元素

如前面所述，获取文档中的元素，其实就是获取 <body> 节点下面的各个元素的过程。获取和定位网页元素的意义是：一旦获取一个元素，就可以对其进行读取和更改。由于所有元素都位于 HTML 文档的树形结构中，因此，理论上说，可以定位到文档中的任意一个元素。

获取和定位元素的方法主要分为以下两个体系。

1. 根据元素的相对位置关系

- ❑ all.tags：返回当前节点中包含的指定名称的后代节点集合。
- ❑ ParentNode：返回当前节点的父节点。
- ❑ ChildNodes：返回当前节点包含的所有子节点集合。
- ❑ FirstChild：返回当前节点的首个子节点。
- ❑ PreviousSlibing：返回当前节点的前一个同级节点。

- ❑ NextSlibing：返回当前节点的后一个同级节点。

下面的程序还是以读取“刘白梦的简历.html”文件为例，程序的思路是获取所有表格，然后以第 2 个表格（对应于网页源代码中的第 36 ~ 41 行）为中心对象，分别获取该对象的上级、下级、同级兄弟的各个对象，打印每个对象的节点名称。

```
Sub 根据相对位置定位元素 ()
    Dim HDoc As MSHTML.HTMLDocument
    Dim tables As MSHTML.IHTMLElementCollection
    Dim table As MSHTML.HTMLTable
'获取网页源代码并赋给 HDoc 的代码略
    Set tables = HDoc.all.tags("table") '文档中的所有 table
    Set table = tables.Item(1) '第 2 个 table
    Debug.Print "父节点：", table.ParentNode.nodeName
    Debug.Print "第 1 个子节点：", table.ChildNodes(0).nodeName
    Debug.Print "第 1 个子节点：", table.FirstChild.nodeName
    Debug.Print "前一个兄节点：", table.PreviousSibling.nodeName
    Debug.Print "后一个兄节点：", table.NextSibling.nodeName
End Sub
```

立即窗口中的打印结果如下。

```
父节点：        BODY
第 1 个子节点：  TBODY
第 1 个子节点：  TBODY
前一个兄节点：  P
后一个兄节点：  BR
```

2. 根据元素自身的名称和属性

HTML DOM 提供了如下 4 个常用的获取当前节点的后代节点的方法。

- ❑ getElementById：根据指定的 id 属性获取对应的元素，由于一个网页中不允许使用重复的 id，因此该方法返回的是唯一的元素。
- ❑ getElementsByClassName：根据指定的 class 属性，返回一个元素集合。
- ❑ getElementsByName：根据指定的 name 属性，返回一个元素集合。
- ❑ getElementsByTagName：根据指定的元素名称，返回一个元素集合。

假设一个网页的 <body> 节点中有如下表单。

```
<form>
    <input id="Keyword" name="InputField" type="text" value="VBA" />
    <br />
    <input id="Salary" name="InputField" type="text" value="8800" />
    <br />
    <button id="Search" name="Button1">搜索</button>
</form>
```

可以看出表单中包含两个 input 元素和一个 button 元素。其中第 2 个 input 元素的 id 是 Salary，name 是 InputField，TagName 是 input。

下面的程序利用元素自身属性通过多种方式定位第 2 个 input 元素。

```
Sub 根据自身属性定位元素 ()
    Dim HDoc As MSHTML.HTMLDocument
    Dim Col As MSHTML.IHTMLElementCollection
    Dim body As MSHTML.HTMLBody
    Dim form As MSHTML.HTMLFormElement
    Dim textbox As MSHTML.HTMLInputTextElement
'获取网页源代码并赋给 HDoc 的代码略
    Set body = HDoc.body
    Set form = HDoc.forms.Item(0) 'HTML 文档中的首个表单
    Set textbox = HDoc.getElementById(v:="Salary")
    Set textbox = HDoc.getElementsByName(v:="InputField").Item(1) '第 2 个 input
    Set textbox = body.getElementsByTagName(v:="input").Item(1)   '第 2 个 input
    Set textbox = form.getElementsByTagName(v:="input").Item(1)   '第 2 个 input
    '以下遍历每个名称为 InputField 的元素
    Set Col = HDoc.getElementsByName("InputField")
    For Each textbox In Col
        Debug.Print textbox.getAttribute("value")
    Next textbox
End Sub
```

代码分析：使用以 getElements 开头的方法获取的是元素集合（类型是 IHTML ElementCollection）。

上述代码中连续 4 行以 Set textbox = 开头的代码均指向同一个对象。

以上程序的源代码文件为“实例文档 83.xlsm”。

13.3.4 innerHTML、outerHTML、innerText、outerText 的区别

网页元素有 innerHTML、outerHTML、innerText、outerText 4 个属性，其中以 HTML 结尾的属性用于设置或返回元素的包含标签的网页代码，以 Text 结尾的属性是不含标签的文本。

inner 是内部，outer 是外部。例如，下面这个超链接元素的内部包含一个 h1 标签。

```
<a id="Blog1" href="https://home.cnblogs.com/u/ryueifu-VBA/"><h1 style="color:red">刘永富的博客</h1></a>
```

因此，这个超链接的 outerHTML 是上述全部 HTML 代码，innerHTML 是夹在 a 标签开始标签与结束标签之间的部分，也就是**代码加粗的部分**。

超链接元素的 innerText、outerText 都是“刘永富的博客”。

13.3.5 使用 InsertAdjacent 系列方法插入元素

HTML DOM 中的网页元素有一些以 InsertAdjacent 开头的方法名称。

- InsertAdjacentElement：在指定位置插入新元素。
- InsertAdjacentHTML：在指定位置插入 HTML 代码。
- InsertAdjacentText：在指定位置插入文本内容。

以上 3 个方法都有一个必须规定的 where 参数，该参数的取值可以是以下 4 个字符串常量。

- "BeforeBegin"：当前元素的开始标签之前。
- "AfterBegin"：当前元素的开始标签之后。
- "BeforeEnd"：当前元素的结束标签之前。
- "AfterEnd"：当前元素的结束标签之后。

以下面的超链接元素为例：

```
<a id="cnblogs" href="https://www.cnblogs.com/" target="_blank">博客园</a>
```

第 1 个 < 的前面用 "BeforeBegin" 表示。

第 1 个 > 的后面用 "AfterBegin" 表示。

第 2 个 < 的前面用 "BeforeEnd" 表示。

第 2 个 > 的后面用 "AfterEnd" 表示。

假设单元格 B2 中存储了一些 HTML 代码，可以看出，一个 div 下面包含了两个 a 元素，如图 13-10 所示。



图 13-10　单元格 B2 中的 HTML 代码

下面的程序以 cnblogs 元素为当前元素，依次使用 InsertAdjacent 系列的方法，在该元素的前后插入一些元素和文本。运行该程序，修改后的网页代码显示于单元格 B4 中。

```
Sub 使用InsertAdjacent系列方法()
    Dim HDoc As MSHTML.HTMLDocument
    Set HDoc = New MSHTML.HTMLDocument
    HDoc.body.innerHTML = Range("B2").Value '形成HTML文档
    Dim div1 As MSHTML.HTMLDivElement
    Set div1 = HDoc.getElementById("div1")
    Dim cnblogs As MSHTML.HTMLAnchorElement
    Set cnblogs = HDoc.getElementById("cnblogs")
    Dim cto As MSHTML.HTMLAnchorElement
```

```
        Set cto = HDoc.createElement("a") '创建一个超链接元素
        With cto
            .href = "https://edu.51cto.com/"
            .target = "_blank"
            .innerText = "51CTO学院"
        End With
        Dim logo As MSHTML.HTMLImg
        Set logo = HDoc.createElement("img") '创建一个图片元素
        logo.src = "https://www.cnblogs.com/images/logo_small.gif"
        cnblogs.insertAdjacentElement where:="BeforeBegin", insertedElement:=cto '在cnblogs开始标签前面插入cto超链接
        cnblogs.insertAdjacentElement where:="AfterBegin", insertedElement:=logo '在cnblogs开始标签后面插入LOGO图片
        cnblogs.insertAdjacentText where:="BeforeEnd", Text:="首页" '在cnblogs结束标签前面加入文本
        cnblogs.insertAdjacentHTML where:="AfterEnd", HTML:="<p>此处换行</p>" '在cnblogs结束标签后面插入一个段落
        Range("B4").Value = HDoc.body.innerHTML
    End Sub
```

13.4 Internet Explorer 浏览器对象

在 VBA 编程中，可以调用网页浏览器对象，实现自动操作网页。浏览器对象具有很多的属性、方法和事件。本节通过具体实例讲解浏览器对象的最常用技术。

首先为 VBA 工程添加"Microsoft Internet Controls"外部引用，如图 13-11 所示。

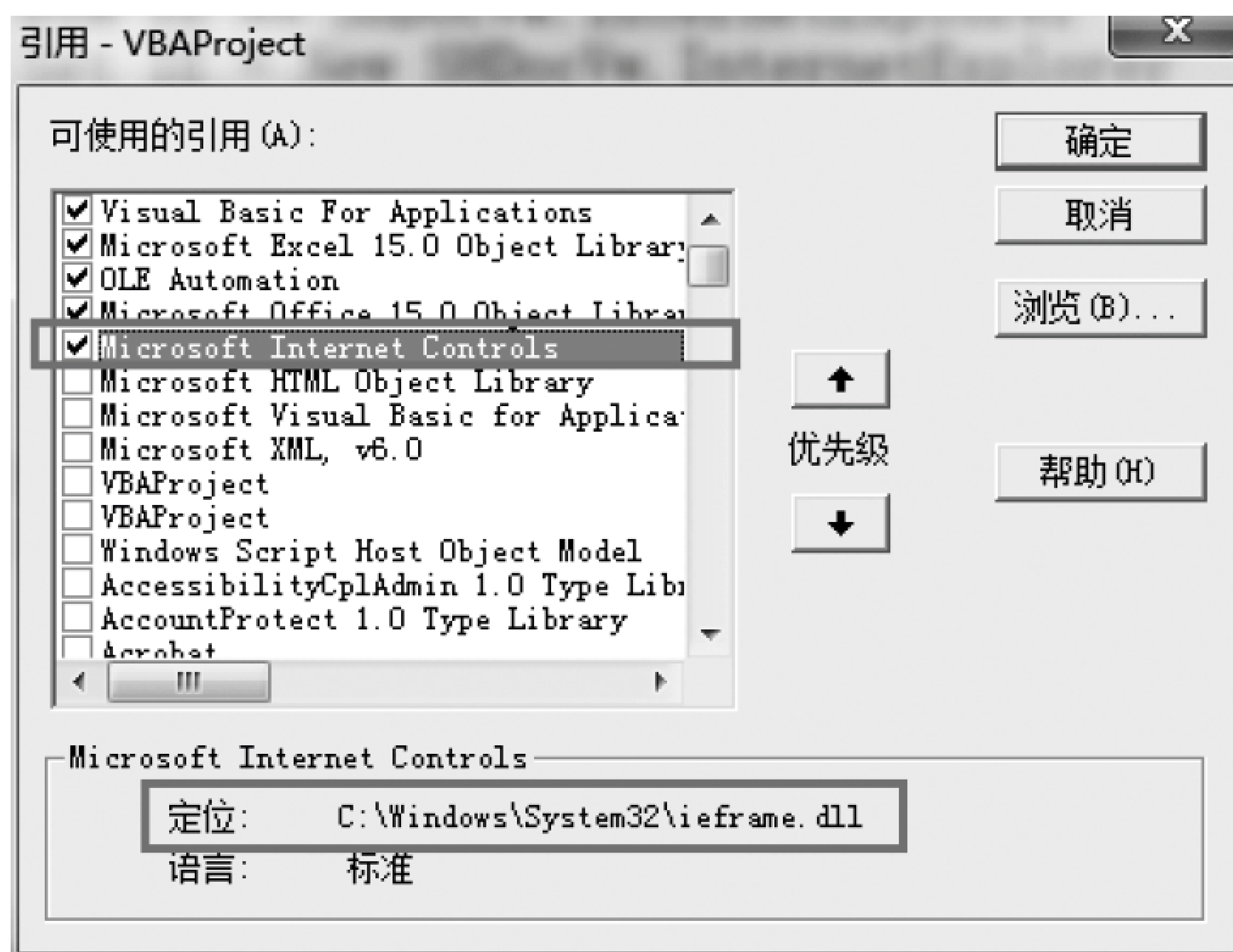


图 13-11 添加外部引用

添加引用后，会在 VBA 工程中引入 SHDocVw 的对象库。

运行下面的程序，会使用默认浏览器打开百度主页。

```
Sub IE对象入门()
    Dim IE As SHDocVw.InternetExplorer
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .Navigate "https://www.baidu.com/"
        MsgBox "下面将自动退出浏览器"
        .Quit
    End With
End Sub
```

代码分析：IE 对象的 Silent=True 表示屏蔽脚本错误，如果不设置这个属性，网页在打开的过程中可能出现脚本错误，如图 13-12 所示。



图 13-12　脚本错误

IE 对象的 Navigate 方法的作用是告诉浏览器打开哪一个 url 的网页。浏览器对象还有一个 Navigate2 方法，该方法还可以用网页浏览本地文件、文件夹，例如 IE.Navigate2 "C:\Source.txt"。

运行上述程序，会在 IE 浏览器中打开百度首页，如图 13-13 所示。



图 13-13　自动启动并且打开网页

本书以 Windows 系统自带的 IE 浏览器讲解，实际应用中也可以把 Firefox、Chrome 等设置为默认浏览器。计算机的默认浏览器的设置存储在如下注册表位置。

```
HKEY_CLASSES_ROOT\http\shell\open\command
```

从图 13-14 可以看出，目前默认的浏览器是搜狗浏览器。

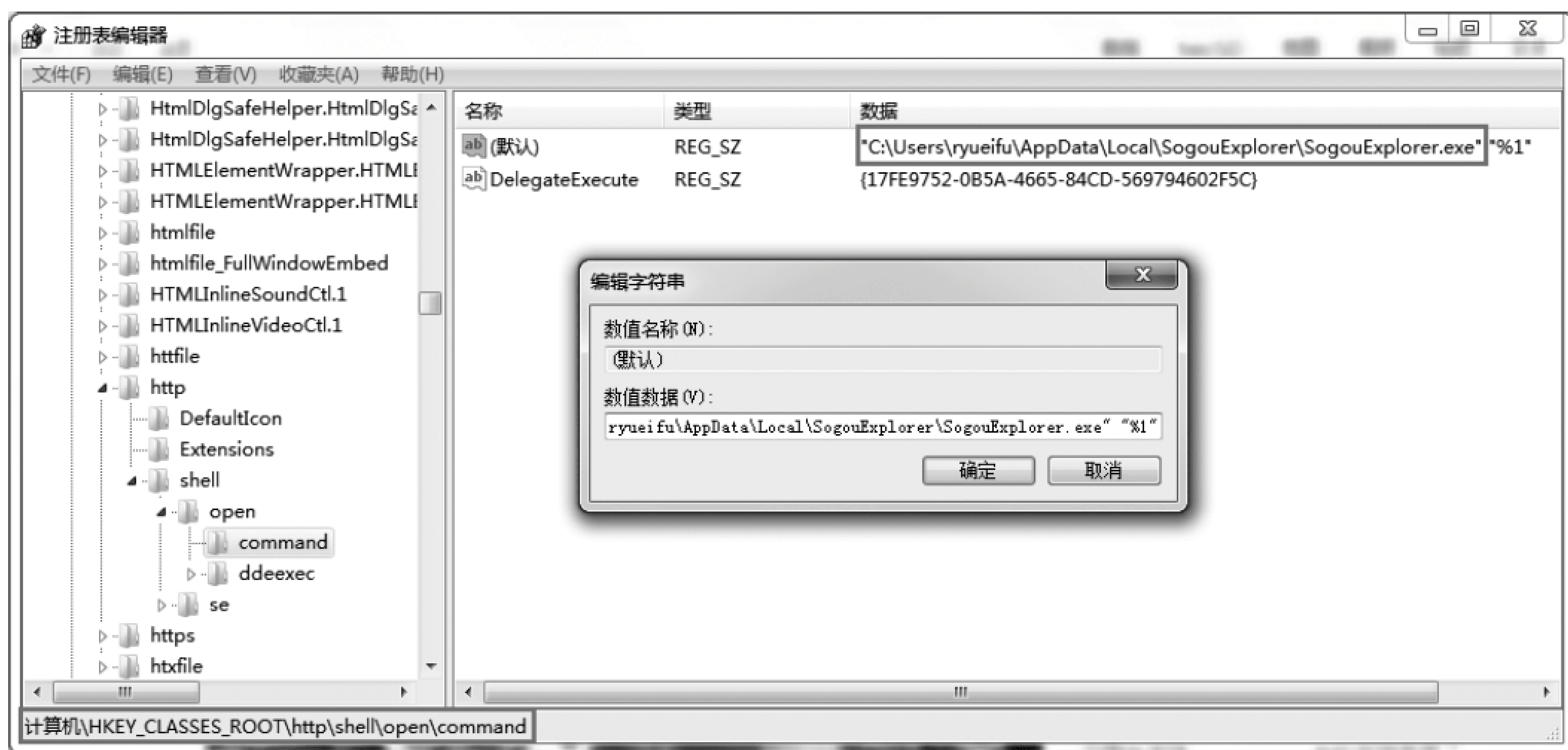


图 13-14　查看和修改默认浏览器

可以手工修改，也可以用 VBA 修改上述注册表项，例如把上述路径修改为 IE 浏览器的程序路径：C:\Program Files\Internet Explorer\iexplore.exe，就可以把 IE 浏览器设置为默认浏览器。

以上是一个最简单的调用浏览器的实例，在实际应用中，可以在浏览器中进行各种操作，例如获取网页数据、自动填写表单、自动单击网页按钮等各种操作。

要想使用浏览器对象对某个网站、网页进行流畅、准确的自动化操作，必须在运行程序前事先分析网页，找出网页中核心元素的各种属性。因此，还需要进行如下两个操作。

- 向工程添加 Microsoft HTML Object Library 的引用。
- 使用浏览器的开发工具分析网页。

其中，使用 Microsoft HTML Object Library 的原因，主要是为了使用 MSHTML 对象库中的对象类型。

13.4.1　使用浏览器的开发工具分析网页元素

网页自动化处理的对象往往是外部网站。网页的源代码是公开的，可以通过查看网页源代码，了解网页的编码方式、网页元素的层次结构、自己感兴趣的数据等信息。

但是，网页源代码往往是一个很庞大的字符串，从里面寻找特定的元素非常费时费力。

大多数的浏览器都有“开发工具”（快捷键都是 F12）的功能，使用开发工具，可以通过单击网页中的对象，快速定位到网页源代码中对应的 HTML 代码，如图 13-15 所示。



图 13-15　打开浏览器的开发工具

例如，在查询快递的网站输入快递单号，可以追踪到物品的配送情况。对于这个查询流程，就可以借助 Internet Explorer 浏览器对象来实现自动化。在写程序之前，首先要用开发工具定位到每个核心元素的 HTML 代码。

对于本案例，需要查看快递单号的输入文本框、右侧的查询按钮，以及下部显示查询结果的元素，如图 13-16 所示。



图 13-16　了解网页构成

在浏览器中按下快捷键 F12，浏览器下部出现一个窗格，在该窗格中切换至“DOM 资源管理器”，如图 13-17 所示。



图 13-17 使用开发工具查看网页元素

然后用鼠标单击 DOM 资源管理器左上角的箭头指针，再把鼠标光标移动到快递单号文本框，下方窗格自动定位到该文本框对应的 HTML 代码，可以看到该文本框是一个表单控件，其 id 是“postid”(如果一个元素有 id 属性，就优先获取 id)。

按照上述操作方法，可以获取到右侧查询按钮的 HTML 代码如下。

```
<a class="btn-default btn-search" id="query"></a>
```

进一步定位快递的查询结果，可以发现查询结果是一个 table 元素，如图 13-18 所示。



图 13-18 查询结果

该 table 元素的 class="result-info"。

掌握了如上 3 处元素的信息，就可以动手写代码了。

```
Sub 自动查询快递信息 ()
    On Error GoTo Err1:
    Dim IE As SHDocVw.InternetExplorer
    Dim textbox As MSHTML.HTMLInputElement
    Dim button As MSHTML.HTMLAnchorElement
    Dim table As MSHTML.HTMLTable
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .Navigate "http://www.kuaidi100.com/"
        While .Busy
            Delay 1000                                         '延时
        Wend
        Set textbox = .Document.getElementById("postid")       '定位快递单号文本框
        Debug.Print textbox.getAttribute("id")
        textbox.Value = "3903861640775"                        '输入一个快递单号
        Set button = .Document.getElementById("query")         '定位查询按钮
        button.Click                                           '单击查询按钮
        While table Is Nothing
            Delay 1000
            Set table = .Document.getElementsByClassName("result-info").Item(0)
                                                               '反复定位查询结果表
        Wend
        Debug.Print table.innerText
        .Quit                                                  '关闭浏览器
    End With
    Exit Sub
Err1:
    Debug.Print Err.Description
    Resume
End Sub
```

代码分析：注意代码中加粗的部分，由于单击了“查询”按钮，查询结果未必能立即刷新出来，所以要放在 While 循环中，只要 table 是 Nothing 就一直执行循环体中的代码，这里运用了错误处理的策略。

其中，延时过程 Delay 的完整代码如下。

```
Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
Sub Delay(interval As Long)
    Dim Savetime As Long
    Savetime = timeGetTime()
    While timeGetTime < Savetime + interval
        DoEvents
    Wend
End Sub
```

运行上述程序，自动启动浏览器，自动输入快递单号，把查询结果打印到立即窗口后，自动退出浏览器。快递查询结果信息如图 13-19 所示。

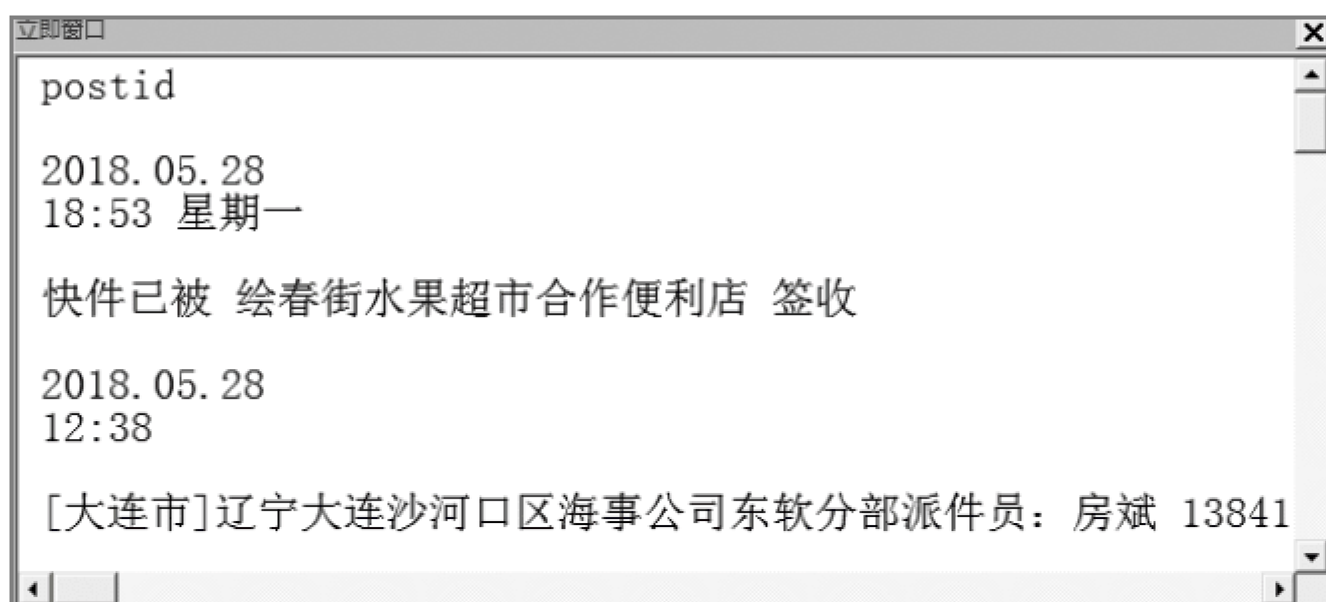


图 13-19　打印 table 元素的 innerText

以上程序的源代码文件为“实例文档 91.xlsm”。

13.4.2　处理超链接弹出的新窗口

网页中往往包含很多超链接，单击一个超链接，有可能在原选项卡显示新的页面，也有可能在右侧出现的新选项卡中显示页面。这主要取决于超链接 a 元素的 target 属性，如果 target 是 _blank，就一定会在新选项卡显示，如果目标页面不在原先选项卡显示，就无法获取目标页面的网页内容，也就是无法操作和控制新选项卡中的内容，为此，要想办法让新页面在原先的选项卡出现。

具体的解决办法有如下 3 个。

- 定位并查看超链接 a 元素的 target 属性，如果 target 是 _blank，把该属性修改为 _self，然后执行超链接元素的 Click 方法，这样就不会在新窗口中弹出。
- 定位并查看超链接 a 元素的 href 属性，如果能找到具体跳转的 url，可以使用 IE 对象直接 Navigate 这个超链接对应的网址。
- 利用浏览器对象的 NewWindow2 事件。

下面通过在百度首页中搜索“SQL 查询前 10 条记录”这个关键字，在多个搜索结果中，自动单击最上面的那个超链接，并查看内容。

首先用开发工具获取重要元素，搜索关键字文本框的 id 是 kw，“百度一下”这个按钮的 id 是 su，如图 13-20 所示。

接下来就可以动手写代码以实现自动搜索关键词。

```
Public IE As SHDocVw.InternetExplorer
Sub 百度搜索 ()
    Dim KeyWord As MSHTML.HTMLInputTextElement
    Dim Search As MSHTML.HTMLInputButtonElement
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Visible = True
        .Silent = True
        .navigate "https://www.baidu.com/"
        While .Busy
            DoEvents
        Wend
```

```
            Set KeyWord = .document.getElementById("kw")
            KeyWord.Value = "SQL 查询前 10 条记录 "
            Set Search = .document.getElementById("su")
            Search.Click
            While .Busy
                DoEvents
            Wend
            Delay 3
            Debug.Print .LocationURL
            .Quit
        End With
    End Sub
```



图 13-20　查看主要元素的属性

运行上述程序，会看到自动启动浏览器，并在百度中搜索指定的关键词，最后，在立即窗口打印浏览器当前的 url。需要注意的是，浏览器对象 Navigate 时的 url 与 LocationURL 不同，因为页面已经变化了，LocationURL 返回的是最新的网址。立即窗口的打印结果如下。

```
    https://www.baidu.com/s?ie=utf-8&f=8&rsv_bp=0&rsv_idx=1&tn=baidu&wd=SQL%E6%9F
%A5%E8%AF%A2%E5%89%8D10%E6%9D%A1%E8%AE%B0%E5%BD%95&rsv_pq=fbf29e7b00004315&rsv_t=
d6aaFAEu50m70X%2FryS3yv8w2g5RSw9%2FOCiFdga%2BQwFHM0Kmpy3YfT5pgoSs&rqlang=cn&rsv_
enter=1&rsv_sug9=eb_1
```

这个搜索网址看起来很长，实际上最核心的部分是 wd=SQL...，其他参数可以忽略，根据这一点，以后就可以用浏览器对象直接 Navigate 带有 wd 参数的网址，而不需要进入百度首页。

因此，上述代码可以改写成如下形式。

```
Sub 直接访问 ()
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Visible = True
```

```
            .Silent = True
             .navigate "https://www.baidu.com/s?" & "wd=" & URLEncode("SQL 查询前 10
条记录 ")
            While .Busy
                DoEvents
            Wend
        End With
    End Sub
```

代码分析：对于网址中包含中文、标点符号，或者发送的数据中包含非英文的内容，就需要先编码，再使用。上述代码中 URLEncode 这个自定义函数稍后讲述。

接下来需要分析的是，对于搜索出的多个结果，如何自动获取到最顶部的那个超链接并且在原页面打开呢？

再次利用开发工具分析，可以发现规律：最上面的搜索结果是一个 div 元素，其 id 是数字 1，下一个搜索结果的 id 是 2。每个 div 中有且仅有一个超链接 a元素，如图 13-21 所示。



图 13-21 搜索结果的 HTML 分析

但是，可以看到超链接 a 元素的 target 是 _blank，这就意味着只要单击这个搜索结果，就会在新选项卡弹出页面。这不是我们期望的。

下面的程序利用自动修改超链接元素的 target 属性为 _self，阻止弹出新窗口。

```
    Sub 在原选项卡打开首个搜索结果 _ 修改 target()
        Dim FirstLink As MSHTML.HTMLAnchorElement
        Set IE = New SHDocVw.InternetExplorer
        With IE
            .Visible = True
            .Silent = True
             .navigate "https://www.baidu.com/s?" & "wd=" & URLEncode("SQL 查询前 10
条记录 ")
            While .Busy
```

```
            DoEvents
        Wend
        Delay 3
        Set FirstDiv = .document.getElementById("1")
        Set FirstLink = FirstDiv.getElementsByTagName("a").Item(0)
        FirstLink.setAttribute strattributename:="target", attributevalue:="_self"
        FirstLink.Click
        Delay 3
        Debug.Print .LocationURL
    End With
End Sub
```

运行上述程序，程序会自动单击第一个搜索结果，在原有选项卡查看其内容，而不弹出新窗口。

此外，定位到超链接元素后，还可以让浏览器对象直接 Navigate 超链接的 href，例如下面这句。

```
IE.navigate FirstLink.getAttribute("href") '或者 FirstLink.href
```

这样也实现了在原选项卡中显示新的页面。

13.4.3　中文字符的编码和解码

由于每个国家都有自己的语言文字，当网址或者提交发送的数据中包含这些字符时，可能无法正常解析，因此，有必要进行字符的编码和解码。

函数 URLEncode 和 URLDecode 就是用于编码和解码的自定义函数，如下所示。

```
'编码函数
Public Function URLEncode(ByRef strURL As String) As String
    Dim i As Long
    Dim tempStr As String
    For i = 1 To Len(strURL)
        If Asc(Mid(strURL, i, 1)) < 0 Then
            tempStr = "%" & Right(CStr(Hex(Asc(Mid(strURL, i, 1)))), 2)
            tempStr = "%" & Left(CStr(Hex(Asc(Mid(strURL, i, 1)))), Len(CStr
(Hex(Asc(Mid(strURL, i, 1)))) - 2) & tempStr
            URLEncode = URLEncode & tempStr
        ElseIf (Asc(Mid(strURL, i, 1)) >= 65 And Asc(Mid(strURL, i, 1)) <= 90)
Or (Asc(Mid(strURL, i, 1)) >= 97 And Asc(Mid(strURL, i, 1)) <= 122) Then
            URLEncode = URLEncode & Mid(strURL, i, 1)
        Else
            URLEncode = URLEncode & "%" & Hex(Asc(Mid(strURL, i, 1)))
        End If
    Next
End Function
'解码函数
Public Function URLDecode(ByRef strURL As String) As String
    Dim i As Long
    If InStr(strURL, "%") = 0 Then URLDecode = strURL: Exit Function
    For i = 1 To Len(strURL)
        If Mid(strURL, i, 1) = "%" Then
            If Val("&H" & Mid(strURL, i + 1, 2)) > 127 Then
```

```
                    URLDecode = URLDecode & Chr(Val("&H" & Mid(strURL, i + 1, 2) & 
Mid(strURL, i + 4, 2)))
                    i = i + 5
                Else
                    URLDecode = URLDecode & Chr(Val("&H" & Mid(strURL, i + 1, 2)))
                    i = i + 2
                End If
            Else
                URLDecode = URLDecode & Mid(strURL, i, 1)
            End If
        Next
    End Function
```

例如：

```
URLEncode("SQL 查询前 10 条记录 ")
```

返回的编码结果如下。

```
SQL%B2%E9%D1%AF%C7%B0%31%30%CC%F5%BC%C7%C2%BC
```

反过来：

```
URLDecode("SQL%B2%E9%D1%AF%C7%B0%31%30%CC%F5%BC%C7%C2%BC")
```

返回的解码结果是：SQL 查询前 10 条记录。

编码和解码函数并非唯一的版本，以上两个是最常用的。

以上程序的源代码文件为“实例文档 92.xlsm”。

13.4.4 使用浏览器对象的事件

Internet Explorer 浏览器对象还支持很多事件过程，如果要使用浏览器的事件，不能在标准模块中声明，而是在类模块中。

在 VBA 工程中插入一个类模块，重命名为 ClassIE，在类模块顶部用 WithEvents 关键字声明一个带有事件过程的浏览器对象。

诸多事件中，NewWindow2 事件可用于处理新窗口的问题，当单击页面中的超链接或者按钮以后，弹出新窗口之前会触发 NewWindow2 事件，如图 13-22 所示。

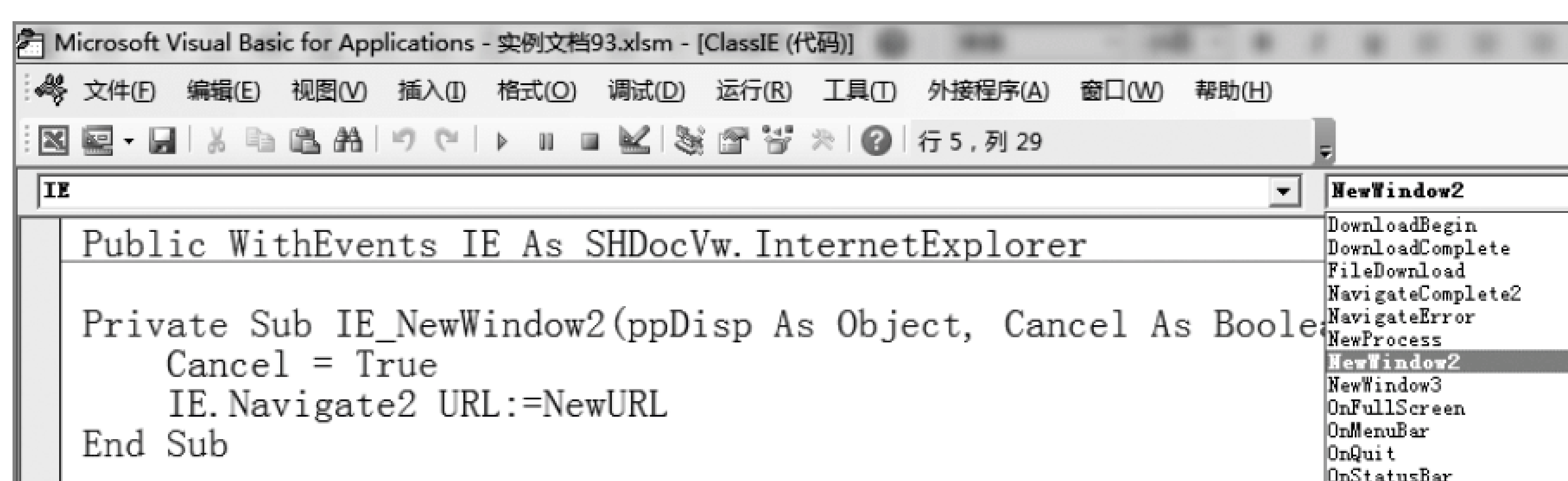


图 13-22 使用 IE 对象的事件

事件代码中的 Cancel=True，表示不弹出新窗口，而是让浏览器 Navigate 新的网址，也

就是跳转到超链接中的网址。

在标准模块中，声明一个类的实例 Instance，以后使用 Instance.IE 作为浏览器。

```
Public Instance As ClassIE
Public NewURL As String
Sub 在原选项卡打开首个搜索结果_使用NewWindow2事件()
    Dim FirstLink As MSHTML.HTMLAnchorElement
    Set Instance = New ClassIE
    Set Instance.IE = New SHDocVw.InternetExplorer
    With Instance.IE
        .Visible = True
        .Silent = True
        .Navigate "https://www.baidu.com/s?" & "wd=" & URLEncode("SQL查询前10
条记录")
        While .Busy
            DoEvents
        Wend
        Delay 3
        Set FirstDiv = .Document.getElementById("1")
        Set FirstLink = FirstDiv.getElementsByTagName("a").Item(0)
        NewURL = FirstLink.href
        FirstLink.Click
        Delay 3
        Debug.Print .LocationURL
    End With
End Sub
```

运行上述程序，也不会产生新窗口，而是在原窗口显示新的页面。

以上程序的源代码文件为“实例文档 93.xlsm”。

13.4.5　处理网页中的表格数据

很多网页上的数据是以表格形式存储的，当需要把网页表格中的数据保存到本地时，可以先定位到 table 元素，然后遍历每行的各个单元格即可。

在 HTML DOM 对象模型中与网页表格有关的对象有如下 3 个。

- ❑ MSHTML.HTMLTable：table 元素。
- ❑ MSHTML.HTMLTableRow：table 中的 tr 元素。
- ❑ MSHTML.HTMLTableCell：tr 中的 td 或 th 元素。

在实际编程中，往往更关心表格中的数据，在获取数据之前，先要了解一个 table 的行数（tr 的个数）和 table 的列数（每行中 td 的个数）。

由于 HTML 表格允许跨行和跨列，也就是合并单元格，从而使得每行的单元格未必都相等。

对于普通的表格，使用 table.rows.length 可以得到表格的行数，table.cells.length 可以得到表格所有的单元格个数，一般情况下，以上两个数字相除就能得到每行单元格的个数，也就是表格的列数。

使用 table.innerText 可以一次性返回表格中的所有文本，如果要存入 Excel 单元格，就

需要使用 VBA 的字符串处理函数了。

使用 row.innerText 可以返回某一行的所有文本。

使用 cell.innerText 可以返回某一个单元格的内容，这个方法最常用。

下面的实例演示了如何自动下载象棋网站中的表格数据。

使用浏览器的开发工具，得到 table 的 id 属性，如图 13-23 所示。



图 13-23 网页表格的 HTML 分析

利用 VBA 定位网页表格，自动把表格内容写到 Excel 单元格中的具体程序如下。

```
Public Sub GetTable()
    Dim IE As SHDocVw.InternetExplorer
    Dim Table As MSHTML.HTMLTable, Row As MSHTML.HTMLTableRow, Cell As MSHTML.
HTMLTableCell
    Dim i As Integer, j As Integer
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .navigate "http://www.gdchess.com/xqgame/xqpgame.asp?pid=0001&res=3"
        While .Busy
            DoEvents
        Wend
        Set Table = .document.getElementById("bpwPlayergame")
        Debug.Print "行数：", Table.Rows.Length
        Debug.Print "单元格数：", Table.Cells.Length
        Debug.Print "第 1 行的列数：", Table.Rows(0).Cells.Length
        Debug.Print "第 3 行 5 列的单元格内容：", Table.Rows(2).Cells(4).innerText
    End With
    '表格内容发送到单元格
    For Each Row In Table.Rows
        i = i + 1
        j = 0
```

```
            For Each Cell In Row.Cells
                j = j + 1
                Sheet1.Cells(i, j).Value = Cell.innerText
            Next Cell
        Next Row
    End Sub
```

代码分析：以上程序大致分为两个部分，一部分用于熟悉表格属性，了解一个表格有多少行多少列。另一部分用于把网页表格数据发送到 Excel 单元格。

注意　Table.Rows(2).Cells(4).innerText 表示第 3 行的第 5 个单元格。HTML DOM 是 0 基的。

运行上述程序，立即窗口打印出这个表格有 81行、10 列，如图 13-24 所示。

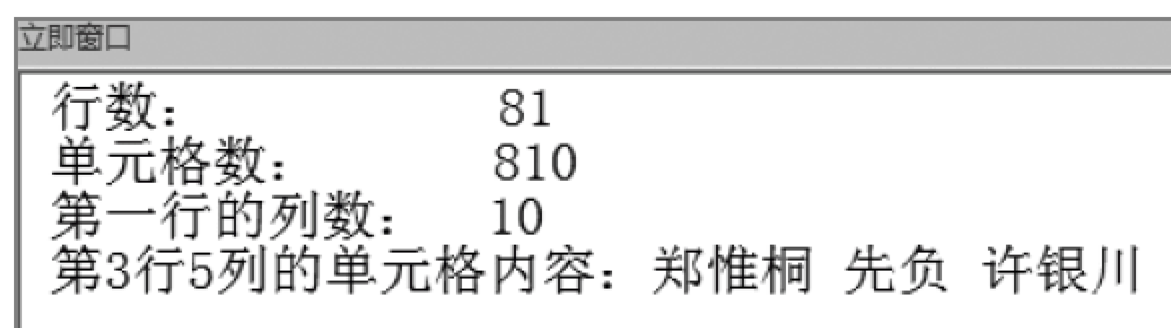


图 13-24　打印网页表格的重要属性

同时，Excel 单元格中获得了网页表格数据，如图 13-25 所示。

实例文档84.xlsm - Excel

文件　开始　插入　页面布局　公式　数据　审阅　视图　开发工具　加载项

A1　序号

	A	B	C	D	E	F	G	H	I	J
1	序号	比赛日期	轮次	台号	对阵棋谱	步数	开局	点击	比赛名称	更新时间
2	1	2018/5/28	3	2	许银川 先胜 金波	47	左中炮对卒	3599	全国象棋男子甲级	2018/5/28 14:53
3	2	2017/12/7	3	2	郑惟桐 先负 许银川	80	左中炮对卒	22674	六届碧桂园杯冠军	2017/12/7 11:33
4	3	2017/12/5	7	4	许银川 先胜 陶汉明	85	对屏风马	22476	第六届碧桂园杯预	2017/12/5 11:39
5	4	2017/12/3	4	2	许银川 先胜 徐超	179	左中炮对卒	12261	第六届碧桂园杯预	2017/12/3 22:22
6	5	2017/12/2	1	2	吕钦 先负 许银川	116	角炮对进左	10677	第六届碧桂园杯预	2017/12/2 12:17
7	6	2017/11/10	29	4	许银川 先胜 蔚强	73	相对左中	15200	全国象棋男子甲级	2017/11/10 12:56
8	7	2017/11/9	28	4	许银川 先胜 徐崇峰	65	相对进３	5754	全国象棋男子甲级	2017/11/9 17:10
9	8	2017/11/5	24	4	许银川 先胜 金松	169	兵互进右马	5225	全国象棋男子甲级	2017/11/5 21:42
10	9	2017/10/15	23	1	许银川 先胜 郭凤达	179	宫炮对横	8218	全国象棋男子甲级	2017/10/16 0:28
11	10	2017/8/12	20	4	曹岩磊 先负 许银川	226	兵互进右马	12213	全国象棋男子甲级	2017/8/12 22:30
12	11	2017/7/29	18	1	许银川 先胜 赵旸鹤	173	左中炮对卒	9542	全国象棋男子甲级	2017/7/29 19:58
13	12	2017/7/23	17	4	许银川 先胜 李智屏	157	人指路对飞	7315	全国象棋男子甲级	2017/7/23 13:07
14	13	2017/7/19	13	4	许银川 先胜 李炳贤	65	左中炮对卒	7421	全国象棋男子甲级	2017/7/20 1:27
15	14	2017/7/18	12	3	许银川 先胜 孙逸阳	191	对屏风马	7026	全国象棋男子甲级	2017/7/19 12:33
16	15	2017/6/25	10	4	柳大华 先负 许银川	90	进七兵对屏	16483	全国象棋男子甲级	2017/6/26 1:44
17	16	2017/5/27	7	4	许银川 先胜 曹岩磊	91	兵互进右马	10865	全国象棋男子甲级	2017/5/27 17:06
18	17	2017/5/7	4	4	许银川 先胜 吴魏	87	人指路对飞	5931	全国象棋男子甲级	2017/5/8 0:04
19	18	2017/5/6	3	3	许银川 先胜 刘俊达	89	相对进左	6923	全国象棋男子甲级	2017/5/7 0:31

图 13-25　网页表格数据发送给 Excel

以上程序的源代码文件为“实例文档 84.xlsm”。

13.4.6　自动读写表单

HTML 语言包含在 <form> 和 </form> 标签中的控件就是表单控件。表单控件一般用 <input> 标签来表示，使用不同的 type 属性表达不同类型的控件。

常用的表单控件有：复选框（checkbox）、标签（label）、文本框（text）、密码框（hidden）、

单选按钮（radio）、按钮（button）等。

另外，还可以用 <select> 元素和 <option> 元素形成下拉框。

很多网页都采用表单控件作为网页服务器和用户进行交互的界面，因此，有必要了解如何自动获取表单控件的状态，如何自动修改表单控件。

例如，下面是网页文件 form1.html 中的 HTML 代码，包含一个表单，该表单中包含一些常用的表单控件。

```
<html>
<head>
    <meta charset="GB2312" />
    <title></title>
</head>
<body>
    <form>
        <input id="checkbox1" type="checkbox" checked="checked" /><label>Excel</label>
        <input id="checkbox2" type="checkbox" /><label>Word</label><br />
        <label>用户：</label><input id="text1" type="text" value=""/>
        <label>密码：</label><input id="password1" type="password" value="" /><br/>
        <input id="radio1" name="熟练程度" type="radio" checked="checked"/>
<label>入门</label>
        <input id="radio2" name="熟练程度" type="radio"/><label>精通</label><br />
        <input id="button1" type="button" value="提交" onclick="javascript:
alert('朋友你好！')" />
    </form>
    <label>学习时长：</label>
    <select id="select1" style="width:100px">
        <option value="One Day">1天</option>
        <option value="One Week" selected="selected">1周</option>
        <option value="One Month">1月</option>
    </select>
</body>
</html>
```

在浏览器中打开该网页文件，如图 13-26 所示。

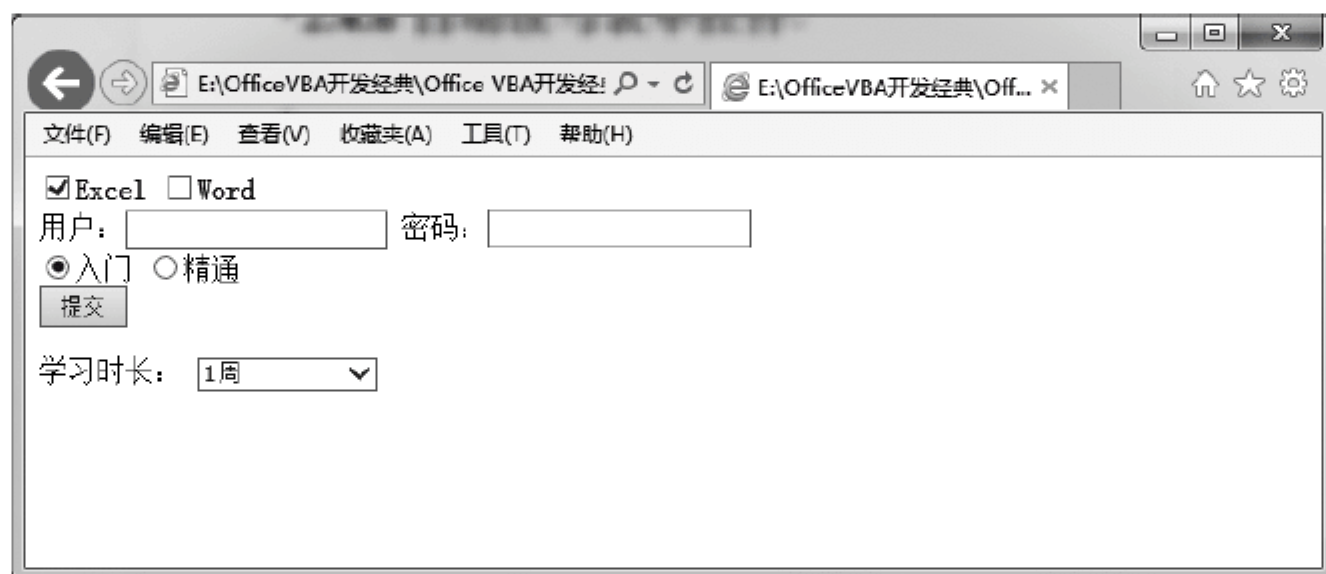


图 13-26　含有表单元素的网页

在使用 HTML DOM 获取和定位表单元素时，需要声明对应元素类型的对象变量，例如文本框元素应该声明为 MSHTML.HTMLInputTextElement 类型。

不同类型的表单控件，读写其属性的方法也有所不同，例如要更改文本框中的内容，可以修改其 Value 属性，而勾选和取消勾选复选框，则要更改其 checked 属性。

```
Sub 自动填写表单 ()
    On Error GoTo Err1:
    Dim IE As SHDocVw.InternetExplorer
    Dim checkbox2 As MSHTML.HTMLInputElement
    Dim username As MSHTML.HTMLInputTextElement, password As MSHTML.HTMLInput
TextElement
    Dim radio2 As MSHTML.HTMLInputElement
    Dim submit As MSHTML.HTMLInputButtonElement
    Dim Options As MSHTML.IHTMLElementCollection
    Set IE = New InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .navigate ThisWorkbook.Path & "\form1.html"
        While .Busy
            DoEvents
        Wend
        Set checkbox2 = .document.getElementById("checkbox2") '定位复选框
        checkbox2.Checked = True                               '勾选复选框
        Set username = .document.getElementById("text1")       '定位"用户名"文本框
        username.Value = "liuyongfu"
        Set password = .document.getElementById("password1")  '定位"密码"文本框
        password.Value = "123456"
        Set radio2 = .document.getElementById("radio2")        '定位单选按钮
        radio2.Checked = True
        Set Options = .document.getElementsByTagName("option")'定位下拉框
        Options.Item(2).Selected = True                        '选中第 3 个
        Set submit = .document.getElementById("button1")       '定位"提交"按钮
        submit.Click
    End With
    Exit Sub
Err1:
    Debug.Print Err.Description
End Sub
```

运行上述程序，自动打开该网页，程序会自动勾选复选框、填写用户名和密码、自动单击“提交”按钮等操作，如图 13-27 所示。

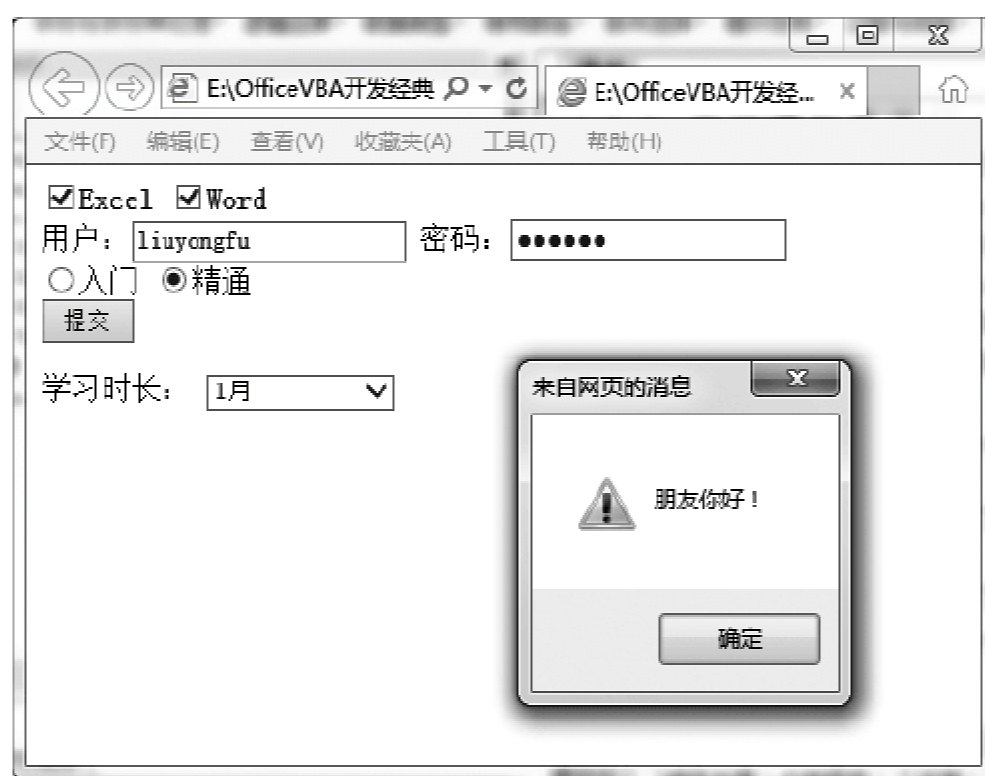


图 13-27　自动填写表单

对于下拉框的自动选择，除了定位 select 下面的 option 以外，还可以直接指定 select 元素的 selectedIndex 属性切换所选项目。具体代码如下。

```
Dim combobox As MSHTML.HTMLSelectElement
Set combobox =IE.document.getElementById
( “select1” ) '定位到下拉框，而不是定位子项
combobox.selectedIndex = 0 '自动选中第 1 项
```

如果表单中有提交按钮（HTML 代码类似于 <input type="submit" value="Submit" />)，除了 Click 这个按钮以外，还可以先定位所属的 form，然后使用 form.submit 方法提交表单。

以上程序的源代码文件为“实例文档 85.xlsm”。

13.5　WebBrowser 控件

WebBrowser 是微软提供的一个用于浏览网页的 ActiveX 控件，该控件的大部分成员、实现原理与上节讲过的 Internet Explorer 是一样的。

WebBrowser 控件与 Internet Explorer 不同的是，前者是植入在程序中的一个控件，网页显示在窗体中，后者是一个对象，网页显示在独立的浏览器中。但它们处理的对象是一样的，都是网页。

WebBrowser 控件可以插入 VBA 的用户窗体以及 VB、C# 等窗体中，还可以插入 Word 文档、Excel 工作表上。一般情况下，VBA 的控件工具箱中找不到该控件，需要在控件工具箱右击，在右键菜单中选择“附加控件”命令，在“附加控件”对话框中，找到“Microsoft Web Browser”并且勾选，如图 13-28 所示。



图 13-28　添加 ActiveX 控件

控件工具箱中出现一个地球形状的控件，这样就可以把该控件拖放到用户窗体中使用。注意，用户窗体中加入 WebBrowser 控件以后，VBA 工程会自动添加“Microsoft Internet Controls”的外部引用，如图 13-29 所示。

用户窗体上插入 WebBrowser 控件之后，就可以使用该控件浏览本地、外部网页，也可以用来显示本地的 XML 文件、gif 图片等。例如：

```
UserForm1.WebBrowser1.Navigate “https://www.baidu.com/”
```

就可以在用户窗体中看到百度首页。

使用 WebBrowser 控件同样可以实现网页自动化，与 Internet Explorer 效果差不多。

本节首先讲解如何访问内嵌于网页中的 iframe 框架里的元素。然后通过自动登录 QQ 邮箱、自动查看收件箱中未读邮件的个数、自动退出登录，讲解 WebBrowser 控件的使用技术。

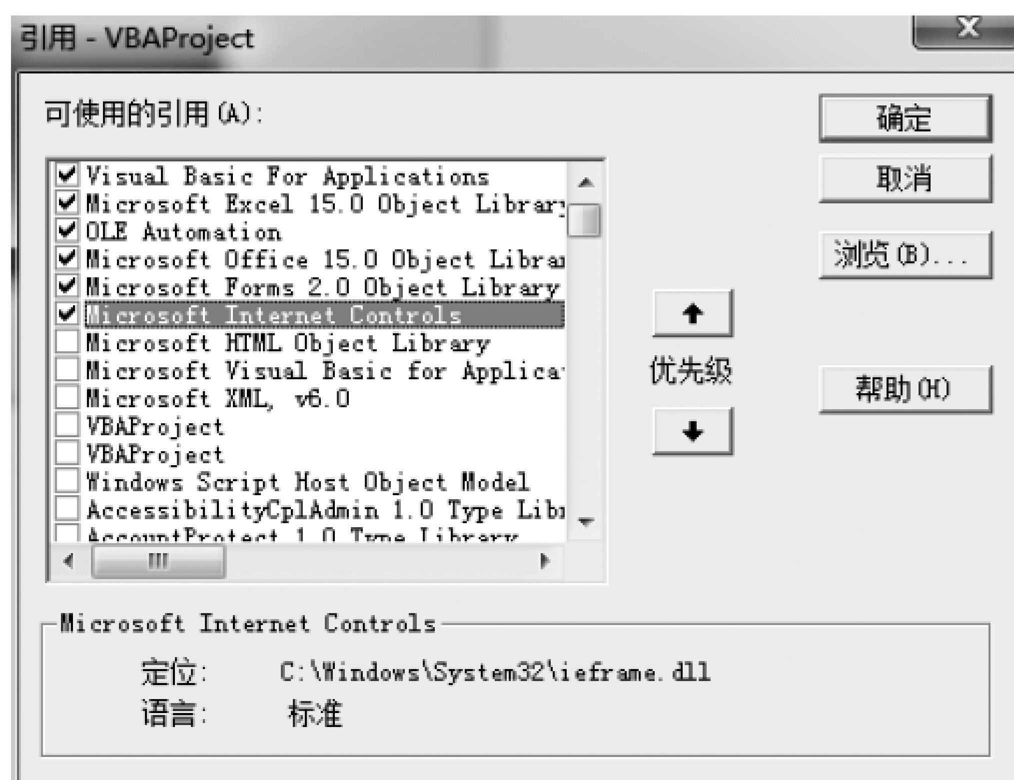


图 13-29　自动添加"Microsoft Internet Controls"外部引用

13.5.1　处理 iframe

iframe 也是 HTML 语言中的一个标签，使用 iframe 框架可以把另一个网页内嵌到主体网页中。iframe 的 src 属性指明了该框架的实际网址。因此，包含 iframe 的网页可以认为是另一个网页寄生在主体网页中。

例如，IP 查询的网址是 http://www.ip138.com/，在 IE 中打开该网页后，在网页中部可以看到本机的 IP 地址和地理位置。

使用浏览器的开发工具来检查元素，会发现 IP 地址和地理位置信息并不在主体网页中，而是位于一个 iframe 的 <center>...</center> 节点中，该 iframe 的 url 为：http://2019.ip138.com/ic.asp，如图 13-30 所示。

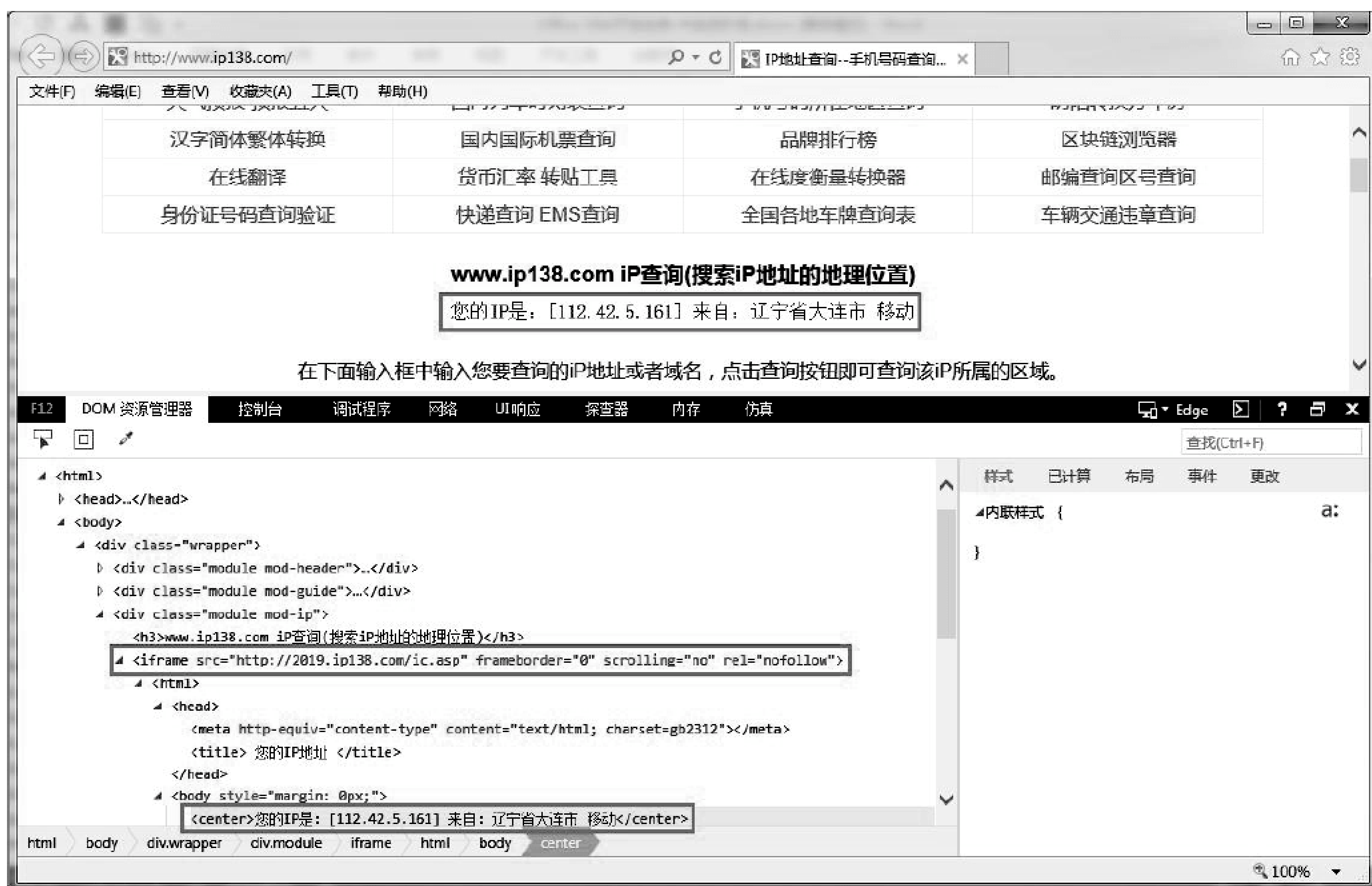


图 13-30　网页中包含 iframe

对于 iframe 框架中的网页元素，可以先从主体网页的 HTML 文档定位到这个 iframe，然后使用 iframe. contentWindow.document 获取框架的 HTML 文档，进而读写框架中的元素。

在用户窗体上放置一个 WebBrowser 控件和一个 CommandButton 控件。命令按钮控件的单击事件代码如下。

```
Private Sub CommandButton1_Click()
    Dim HDoc1 As MSHTML.HTMLDocument
    Dim HDoc2 As MSHTML.HTMLDocument
    Dim iframe As MSHTML.HTMLIFrame
    With Me.WebBrowser1
        .Silent = True
        .Navigate "http://www.ip138.com/"
        While .Busy
            DoEvents
        Wend
        Set HDoc1 = .document ' 取得 WebBrowser 控件的文档，赋给 HDoc1
    End With
    Set iframe = HDoc1.getElementsByTagName("iframe").Item(0) ' 第 1 个框架
    Set HDoc2 = iframe.contentWindow.document
    Debug.Print HDoc2.DocumentElement.outerHTML
End Sub
```

启动用户窗体，单击用户窗体中的命令按钮，WebBrowser 控件首先显示 IP 查询主页的页面内容，然后弹出一个“拒绝的权限”异常对话框，如图 13-31 所示。



图 13-31 在 WebBrowser 控件中显示 iframe 指向的页面

Set HDoc2 = iframe.contentWindow.document 这行代码能否正常执行，与网站有关，有的网站可以正常运行。

如果出现上述异常，可以采用页面跳转的方式，也就是让 WebBrowser 控件 Navigate 一下 iframe 元素的 src 属性中设置的 url。完整代码如下。

```
Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
Private Sub CommandButton1_Click()
    Dim HDoc1 As MSHTML.HTMLDocument
    Dim HDoc2 As MSHTML.HTMLDocument
    Dim Frame As MSHTML.HTMLIFrame
```

```
    Dim center As MSHTML.HTMLObjectElement
    With Me.WebBrowser1
        .Silent = True
        .Navigate "http://www.ip138.com/"
        While .Busy
            DoEvents
        Wend
        Delay 3000
        Set HDoc1 = .document '主页的文档
        Set Frame = HDoc1.getElementsByTagName("iframe").Item(0)
        .Navigate Frame.getAttribute("src") '跳转到 iframe 指定的 url
        While .Busy
            DoEvents
        Wend
        Delay 3000
        Set HDoc2 = .document '此处就是框架的文档
        Set center = HDoc2.getElementsByTagName("center").Item(0)
        MsgBox center.innerText, vbInformation
    End With
End Sub

Private Sub WebBrowser1_NewWindow2(ppDisp As Object, Cancel As Boolean)
    Cancel = True
    Me.WebBrowser1.Navigate2 Me.WebBrowser1.document.activeElement.href
End Sub

Sub Delay(interval As Long)
    Dim Savetime As Long
    Savetime = timeGetTime()
    While timeGetTime < Savetime + interval
        DoEvents
    Wend
End Sub
```

以上程序中，首先打开主页，然后从主页文档中查找到 iframe 跳转的 url，3 秒后在原浏览器控件中继续打开这个 url，进而查找到 center 元素。

运行上述程序，WebBrowser 控件最后显示的是框架页面内容，并且弹出显示结果的对话框，如图 13-32 所示。

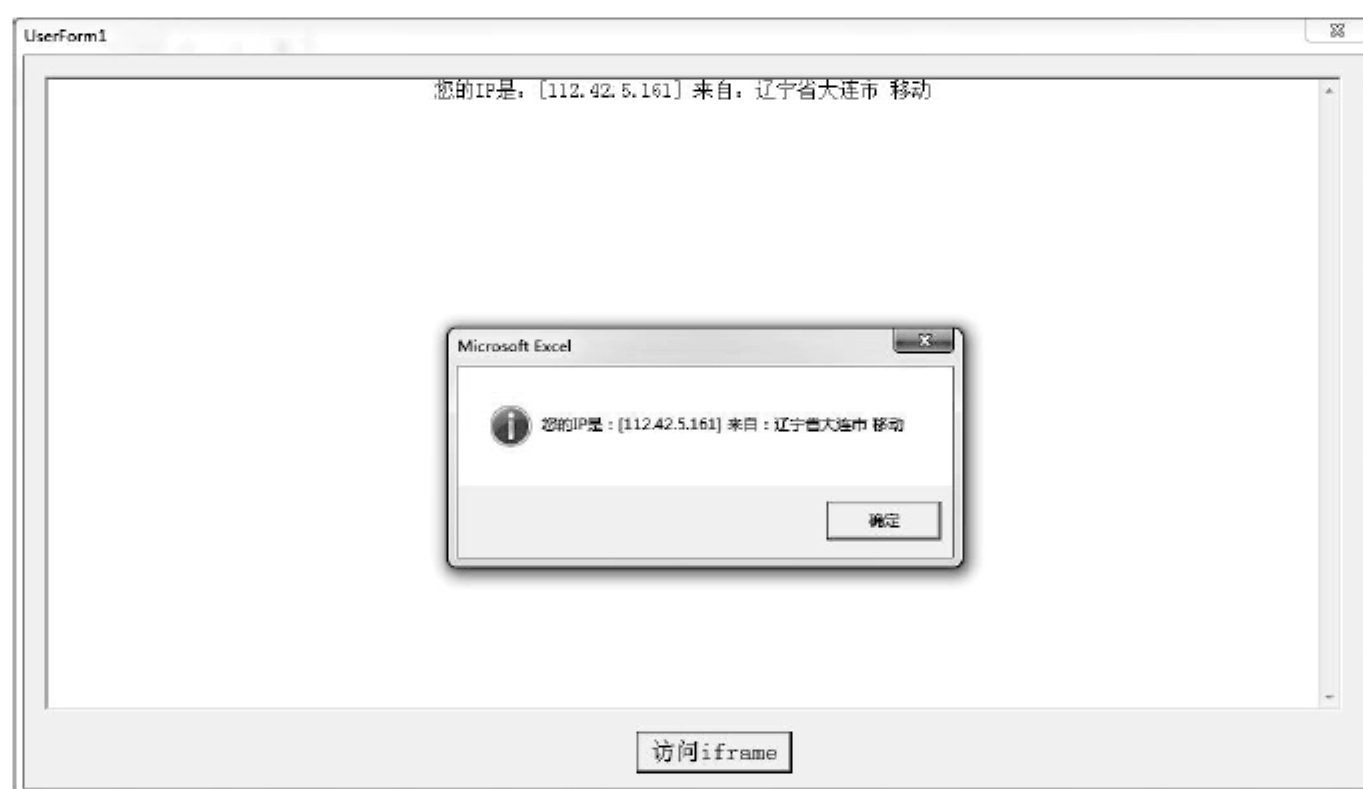


图 13-32　打印 iframe 中的内容

另外，还有一类 frame 框架，访问这类框架文档的方式是：Set HDoc=frame.contentDocument。

以上程序的源代码文件为“实例文档 86.xlsm”。

13.5.2 自动查看邮箱信息

在网页自动化的过程中，经常遇到涉及账户登录的网页，下面这个实例实现了在 WebBrowser 控件中自动登录邮箱、查看未读邮件个数、退出邮箱等操作。

通过开发工具检查邮箱登录页面，发现用户名和密码输入框处于一个 iframe 里面，如图 13-33 所示。

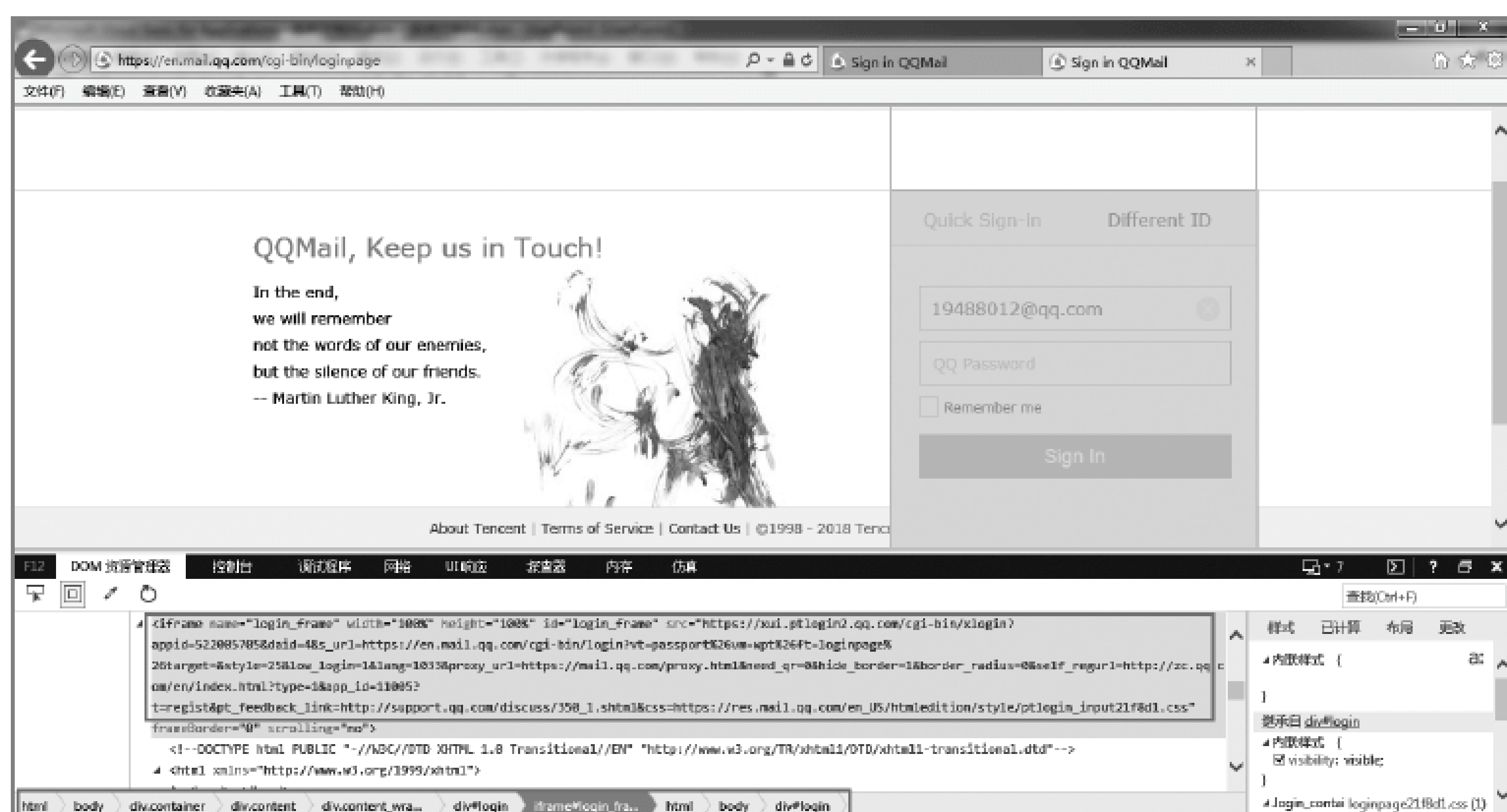


图 13-33　用户名和密码位于 iframe 中

这个 iframe 的 id 是 login_frame，因此非常容易定位，如果复制 iframe 的 src 属性，粘贴到浏览器地址栏中并按回车键，可以清楚地看到登录界面，如图 13-34 所示。



图 13-34　直接浏览 iframe 的 src

有以上的基础研究为依据，就形成了如下的开发思路和流程。

Step 1：WebBrowser 控件 Navigate 邮箱登录的主页。

Step 2：定位 iframe，并且 Navigate 这个 iframe 的 src。

Step 3：切换到选项卡“Different ID”，使得用户名和密码输入框可见。

Step 4：定位用户名和密码输入框，并且自动输入账号信息。

Step 5：自动单击“Sign In”按钮，登录邮箱。

Step 6：进入邮箱，定位并获取页面左侧的“Inbox”。

Step 7：获取到未读邮件个数后，定位并获取页面右上角的“Sign Out”，退出登录。

具体程序代码如下。

```
Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
Private Sub CommandButton1_Click()
    Dim Frame As MSHTML.HTMLIFrame
    Dim ChangeUser As MSHTML.HTMLAnchorElement
    Dim UserName As MSHTML.HTMLInputTextElement
    Dim Password As MSHTML.HTMLInputTextElement
    Dim Login As MSHTML.HTMLInputButtonElement
    Dim Inbox As MSHTML.HTMLAnchorElement
    Dim Links As MSHTML.HTMLElementCollection
    Dim link As MSHTML.HTMLAnchorElement
    With Me.WebBrowser1
        .Silent = True
        .Navigate "https://en.mail.qq.com/cgi-bin/loginpage"
        While .ReadyState <> READYSTATE_COMPLETE
            DoEvents
        Wend
        Delay 3000
        Set Frame = .Document.getElementById("login_frame")
        .Navigate Frame.getAttribute("src")
        While .ReadyState <> READYSTATE_COMPLETE
            DoEvents
        Wend
        Delay 3000
        Set ChangeUser = .Document.getElementById("switcher_plogin")
        ChangeUser.Click
        Delay 3000
        Set UserName = .Document.getElementById("u")
        UserName.Value = ""
        Delay 3000
        UserName.Value = "19488012@qq.com"
        Set Password = .Document.getElementById("p")
        Password.setAttribute strAttributeName:="value", AttributeValue:="123456"
        Set Login = .Document.getElementById("login_button")
        Login.Click
        Delay 3000
        Set Inbox = .Document.getElementById("folder_1")
        MsgBox "未读邮件: " & Inbox.getAttribute("title")
        Set Obj = .Document.getElementsByTagName("a")
        For Each link In Obj
            Debug.Print link.innerText
            If link.innerText = "Sign out" Then
                .Navigate link.getAttribute("href")
                Exit For
            End If
        Next link
    End With
End Sub

Private Sub WebBrowser1_NewWindow2(ppDisp As Object, Cancel As Boolean)
```

```
    Cancel = True
    Me.WebBrowser1.Navigate2 Me.WebBrowser1.Document.activeElement.href
End Sub

Sub Delay(interval As Long)
    Dim Savetime As Long
    Savetime = timeGetTime()
    While timeGetTime < Savetime + interval
        DoEvents
    Wend
End Sub
```

代码分析：WebBrowser 控件有很多可用的事件过程，本实例使用了 NewWindow2 事件，以防止弹出新窗口。

启动用户窗体，单击“登录 QQ 邮箱”按钮，可以看到 WebBrowser 控件中自动登录邮箱，并且对话框弹出未读邮件的个数，如图 13-35 所示。

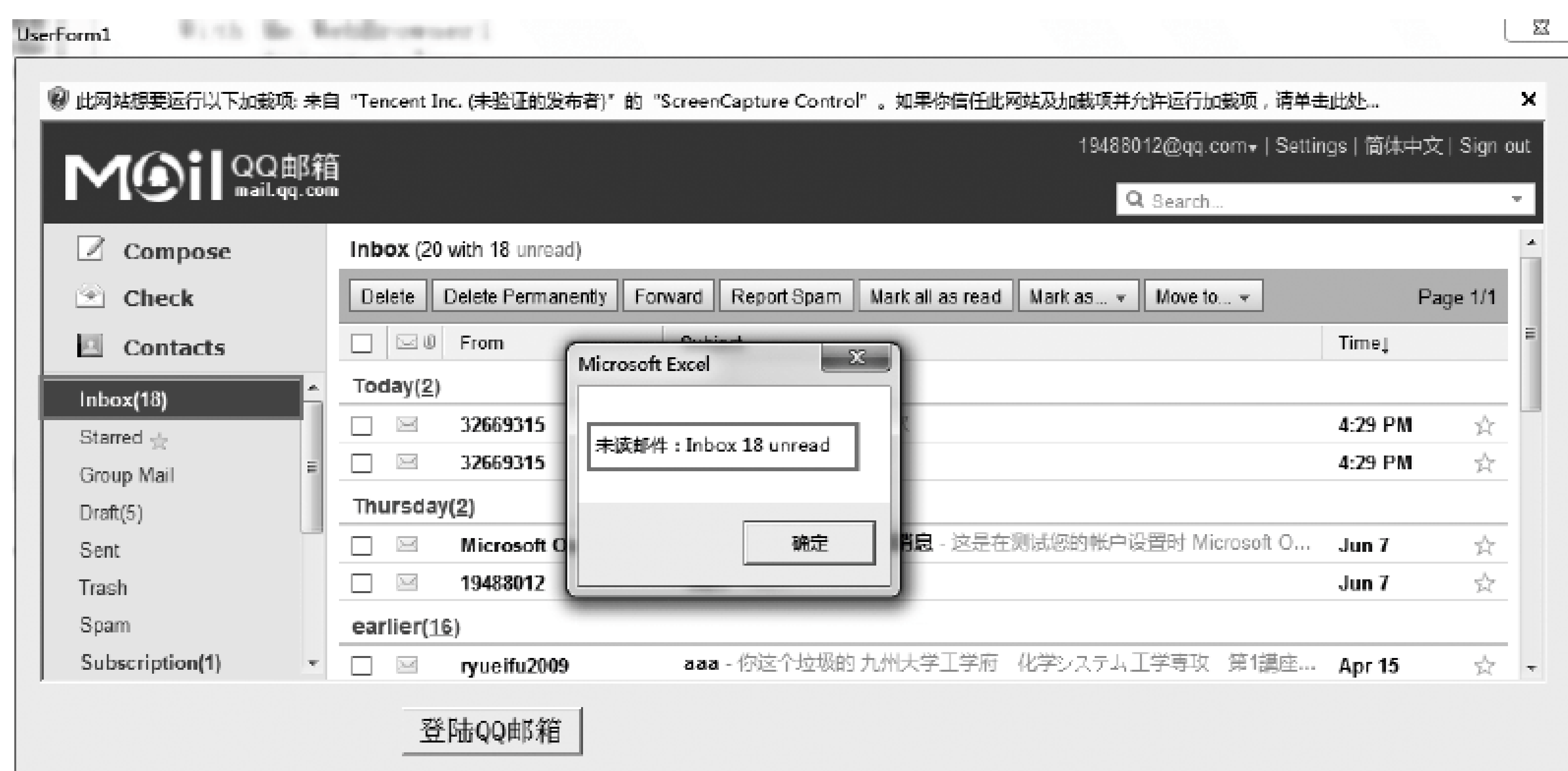


图 13-35　自动登录邮箱

手工关闭 MsgBox 对话框后，自动退出邮箱，如图 13-36 所示。



图 13-36　自动退出登录

以上程序的源代码文件为“实例文档 94.xlsm”。

13.5.3　延时等待处理

Internet Explorer 浏览器对象和 WebBrowser 控件引用的是同一个对象库，能够处理的网页自动化业务大致相同，其实都是浏览器对象。

调用浏览器对象的 Navigate、Refresh 方法，或者单击页面中的按钮引起页面的变化，在页面变化尚未完成之前，不能访问浏览器的文档对象。因此，在使用这些方法之后，一般要加入一定量的延时，等待页面完全就绪后再对其操作。

可以采用的延时策略分为灵活延时和固定延时两类。

灵活延时是在循环体中一直判断浏览器的 readystate 是否等于 4，或者判断 Busy 属性是不是为 True。这两个条件既可以单独使用，也可以与 Or 组合使用。具体特点是页面没有刷新出来或者处于忙碌状态，代码处于循环体中，页面一旦就绪，就跳出循环。总体的等待时间取决于网速、网页的打开速度。

```
Sub 等待就绪()
    Dim IE As SHDocVw.InternetExplorer
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .navigate "http://www.gdchess.com/"
        While .readyState <> READYSTATE_COMPLETE Or .Busy
            DoEvents
        Wend
        Debug.Print .Document.body.innerhtml
    End With
End Sub
```

代码分析：上述程序中，斜体部分就是延时等待处理，如果去掉或注释掉这 3 行代码，再次运行会造成“自动化错误”，如图 13-37 所示。

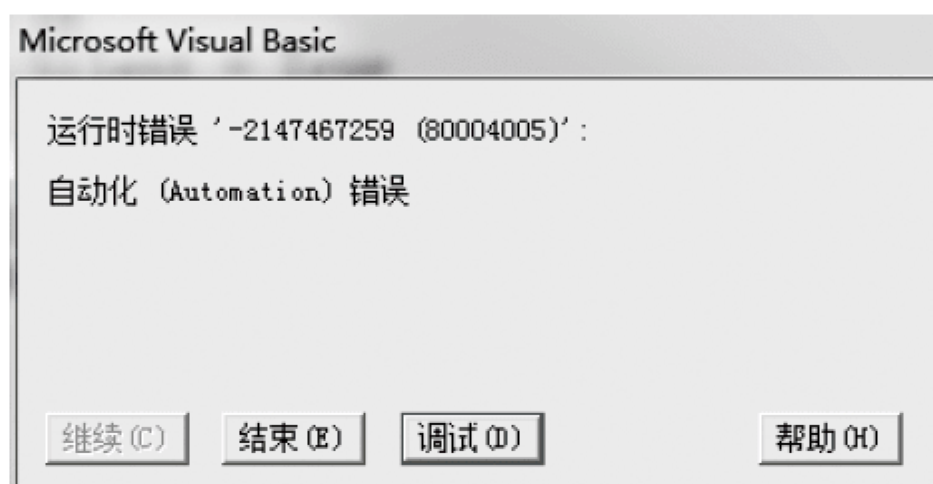


图 13-37　不延时造成的错误

注意　以上所述的灵活延时未必在任何情况都好用，很多情况下使用了上述延时技术经常导致如下两种异常。

- ❑ 程序代码滞留在循环体中出不去，造成死循环（While 后面的条件一直是 False）。
- ❑ 延时之后，使用 GetElement 之类的方法获取不到页面上的元素。

如果灵活延时不能很好地解决问题，可以考虑固定延时。常用的固定延时写法如下。

```
Private Declare PtrSafe Function timeGetTime Lib "winmm.dll" () As Long
Sub Delay1(second As Integer)
    Application.Wait Now + TimeValue("00:00:0" & second)
End Sub
Sub Delay2(millisecond As Long)
    Dim Savetime As Long
    Savetime = timeGetTime()
    While timeGetTime < Savetime + millisecond
        DoEvents
    Wend
End Sub
```

上述代码中过程 Delay1 的延时单位是秒，Delay2 的延时单位是毫秒。例如，Delay1 10 可以延时 10 秒，Delay2 3000 可以延时 3 秒，把这些语句插入浏览器的 Navigate 方法之后即可。

固定延时技术可以自行指定等待时间长短，但是缺点很明显：不知道浏览器的状态。

无论是哪一种延时技术，根本原理都是在无限循环体中使用 DoEvents 转让控制权。

13.5.4 确保元素的获取

浏览器中打开页面时，网页上的各个元素不是同时出现的，用 GetElement 之类的方法获取页面中尚未出现的元素时就会导致错误。有时即使同时使用了灵活延时和固定延时，还是不能确保获取元素。

对于使用以 getElements 开头的方法返回元素集合的场合，判断其是不是 Nothing，如果是 Nothing，就在循环体中一边延时一边获取，直至不是 Nothing 为止。

例如，下面的程序在循环体中一直获取一个框架。

```
Sub 确保集合中的一个元素出现()
    Dim IE As SHDocVw.InternetExplorer
    Dim frame As MSHTML.HTMLFrameElement
    Set IE = New SHDocVw.InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .navigate "http://www.gdchess.com/"
        While .Busy
            DoEvents
        Wend
        Set frame = Nothing
        Do
            Set frame = .Document.getElementsByTagName("iframe").Item(0)
            DoEvents
            If frame Is Nothing = False Then Exit Do
        Loop
        Debug.Print frame.outerHTML
    End With
End Sub
```

对于使用 getElementById 获取元素的场合，使用 IsNull 来判断是不是已经获取到，如果获取到就退出循环。

下面的程序在页面中不断获取一个指定 id 属性的 div 元素。

```
Sub 确保具有 id 的元素出现 ()
    Dim IE As SHDocVw.InternetExplorer
    Dim div As MSHTML.HTMLDivElement
    Set IE = New InternetExplorer
    With IE
        .Silent = True
        .Visible = True
        .navigate "http://www.w3school.com.cn/h.asp"
        While .Busy
            DoEvents
        Wend
        Set div = Nothing
        Do While IsNull(.Document.getElementById("maincontent"))
            DoEvents
        Loop
        Set div = .Document.getElementById("maincontent")
        Debug.Print div.outerHTML
    End With
End Sub
```

使用了上述策略，只有当要找的目标元素顺利获取到才继续运行后面的代码，从而增强程序的健壮性。

以上程序的源代码文件为“实例文档 88.xlsm”。

13.5.5　获取和操作已经打开的浏览器网页

一般情况下，使用 New、CreateObject 创建的浏览器对象（WebBrowser、Internet Explorer），在程序代码中具有持久控制权，可以读写浏览器对象的各方面。

实际上，对于桌面上已经存在的浏览器网页，也可以用程序代码获取并进一步对其操作控制。

如果 VBA 工程中已经添加了“Microsoft Internet Controls”的外部引用，使用 SHDocVw.ShellWindows 可以获取已经打开的所有浏览器网页、所有文件资源管理器窗口。

假设在 IE 浏览器中打开了两个网页，并且开启了两个文件资源管理器，如图 13-38 所示。

下面的程序可以遍历各个窗口的信息到 Excel 单元格中。

```
Public Browser As SHDocVw.WebBrowser
Public Sub 遍历所有浏览器窗口 ()
    Dim AllWindows As SHDocVw.ShellWindows
    Set AllWindows = New SHDocVw.ShellWindows
    Dim i As Integer
    i = 1
    Range("A" & i & ":E" & i).Value = Array(" 名称 ", " 类型 ", " 标题 ", " 网址 / 路径 ", " 句柄 ")
    For Each Browser In AllWindows
        i = i + 1
        Range("A" & i & ":E" & i).Value = Array(Browser.Name, TypeName(Browser.
```

```
document), Browser.LocationName, Browser.LocationURL, Browser.Hwnd)
        Next Browser
    End Sub
```

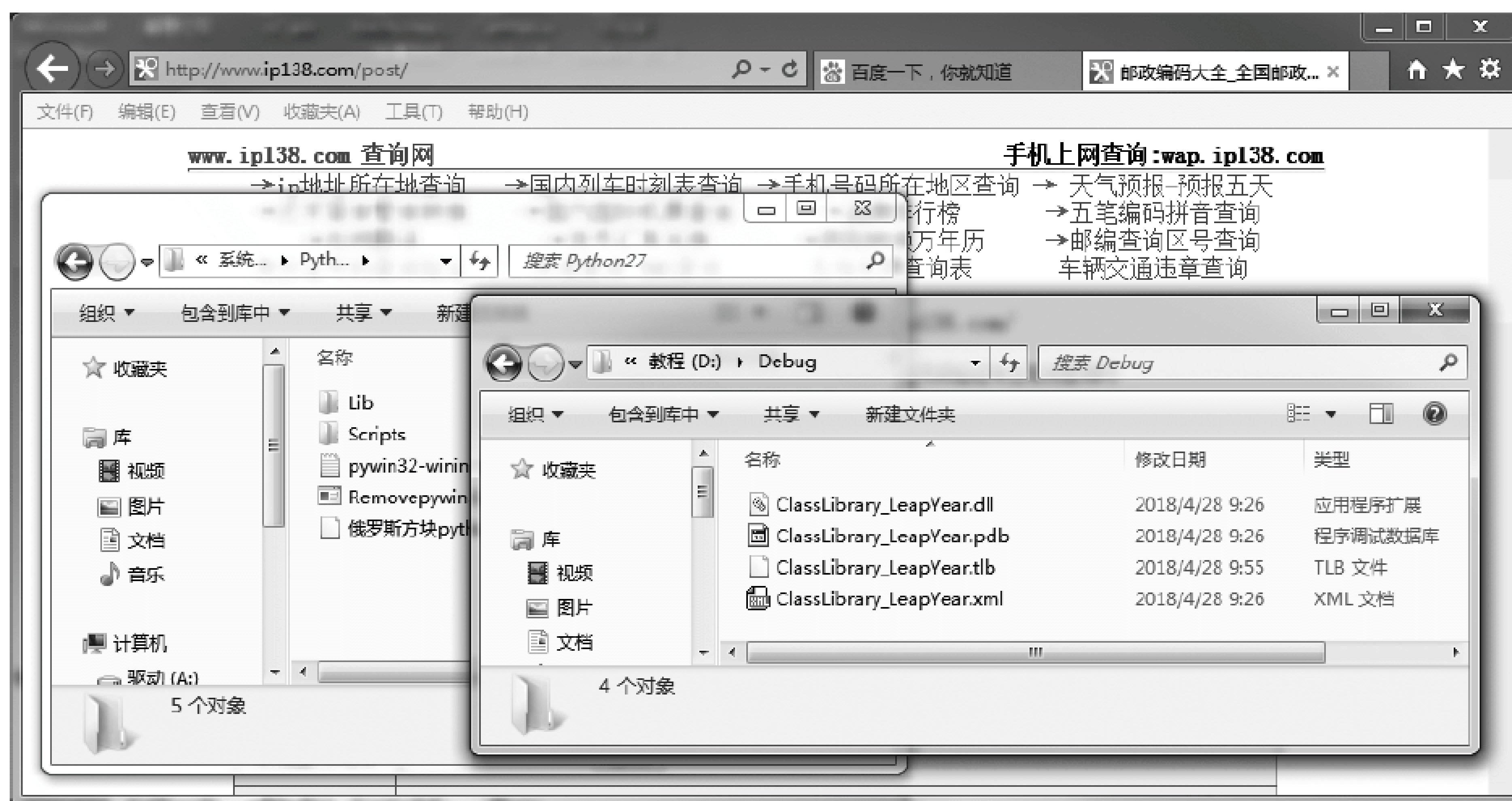


图 13-38　事先打开若干窗口

代码分析：浏览器网页窗口的 Name 属性、Browser.document 类型，与文件资源管理器窗口不同，另外，文件资源管理器的 LocationURL 属性是以 file:// 开头的，根据这些特点就可以区分开哪些是网页窗口，哪些是资源管理器窗口。

运行上面的程序，单元格中列出各个窗口的 5 个比较重要的属性，如图 13-39 所示。

	A	B	C	D	E
1	名称	类型	标题	网址/路径	句柄
2	Windows 资源管理器	IShellFolderViewDual3	Python27	file:///C:/Python27	328190
3	Internet Explorer	HTMLDocument	百度一下，你就知道	https://www.baidu.com/	8652242
4	Internet Explorer	HTMLDocument	邮政编码大全_全国	http://www.ip138.com/post/	8652242
5	Windows 资源管理器	IShellFolderViewDual3	Debug	file:///D:/Debug	657266

图 13-39　窗口对象的重要属性

这里强调一下句柄值，因为多个网页标签可以共用一个浏览器窗口，所以句柄是一样的。而资源管理器是相对独立的窗口，所以句柄值不同。上述程序中的变量 Browser 也可以声明为 Internet Explorer，它和 WebBrowser 对象的成员相同。

既然可以获取到提前打开的网页，如何对某一个特定的网页进行操作和控制呢？判断的技巧就是根据这个网页与其他网页属性的不同。

假设桌面上已经打开了多个网页，下面的程序可以把“百度一下”的那个网页赋给对象变量 Browser，从而对其单独操作。

```
Public Sub 操作已经打开的浏览器网页 ()
    Dim AllWindows As SHDocVw.ShellWindows
    Set AllWindows = New SHDocVw.ShellWindows
    Dim H As New HTMLDocument
    For Each Browser In AllWindows
```

```
        If Browser.Application = "Internet Explorer" Then
            If Browser.LocationURL Like "https://www.baidu.com/*" Then Exit For
        End If
    Next Browser

    With Browser
        .MenuBar = False                '隐藏菜单栏
        .StatusBar = True               '显示浏览器的状态栏
        .statusText = "程序正在运行..."
        Delay 2
        Set H = .document
        H.getElementById("kw").Value = "VBA 实现网页自动化"
        H.getElementById("su").Click
        Delay 2
        .Refresh                        '刷新网页
        Delay 2
        .GoBack                         '后退
        Delay 2
        .Stop                           '停止
        Delay 2
        .Quit                           '退出
    End With
End Sub
```

代码分析：在 For 循环中遍历各个窗口时，第一层 If 判断用于过滤文件资源管理器窗口，第二层 If 判断是找出 url 中包含指定网址的网页，如果找到就跳出 For 循环，之后，就可以在程序代码中通过读写 Browser 这个对象变量来操控实际存在的网页了。

运行上述程序，隐藏浏览器的菜单栏，显示状态栏，并设定状态栏中的文字。然后定位各个有关网页元素输入内容并执行搜索，如图 13-40 所示。



图 13-40　获取和操作正在运行的 IE 对象

以上技术对于网页自动化程序的调试带来很大方便，因为不需要从头创建浏览器，从网页中间的任何一个画面和阶段都可以读写现存的网页。

13.5.6　获取和操作文件资源管理器窗口

系统的文件资源管理器窗口也是一种 WebBrowser 对象，同样可以实现自动化。

手工打开多个资源管理器窗口，运行下面的程序，可以把特定的一个窗口赋给对象变量 Browser，进一步更改窗口的大小，改变浏览器的路径，自动前进和后退，自动退出。

```
Sub 自动操作资源管理器()
    On Error GoTo Err1:
    Dim AllWindows As SHDocVw.ShellWindows
    Delay 2
    Set AllWindows = New SHDocVw.ShellWindows
    For Each Browser In AllWindows
        If Browser.Application = "Windows 资源管理器" Then
            If Browser.LocationName = "Python27" Then Exit For
        End If
    Next Browser

    With Browser
        .AddressBar = False                     '隐藏地址栏
        .FullScreen = False                     '不使用全屏
        .Left = 200
        .Top = 200
        .Width = 1000
        .Height = 500
        .resizable = False                      '不可更改窗口大小
        Delay 2
        .navigate "E:\CDOSendMail"              '变更路径
        While .readyState <> READYSTATE_COMPLETE
            DoEvents
        Wend
        .navigate "D:\Debug"
        Delay 2
        .GoBack
        Delay 2
        .Refresh
        Debug.Print .LocationName, .LocationURL
        Delay 2
        .Quit
    End With
    Exit Sub
Err1:
    Debug.Print Err.Description
End Sub
```

运行上述程序，资源管理器窗口自动更改为指定的大小和位置，如图 13-41 所示。

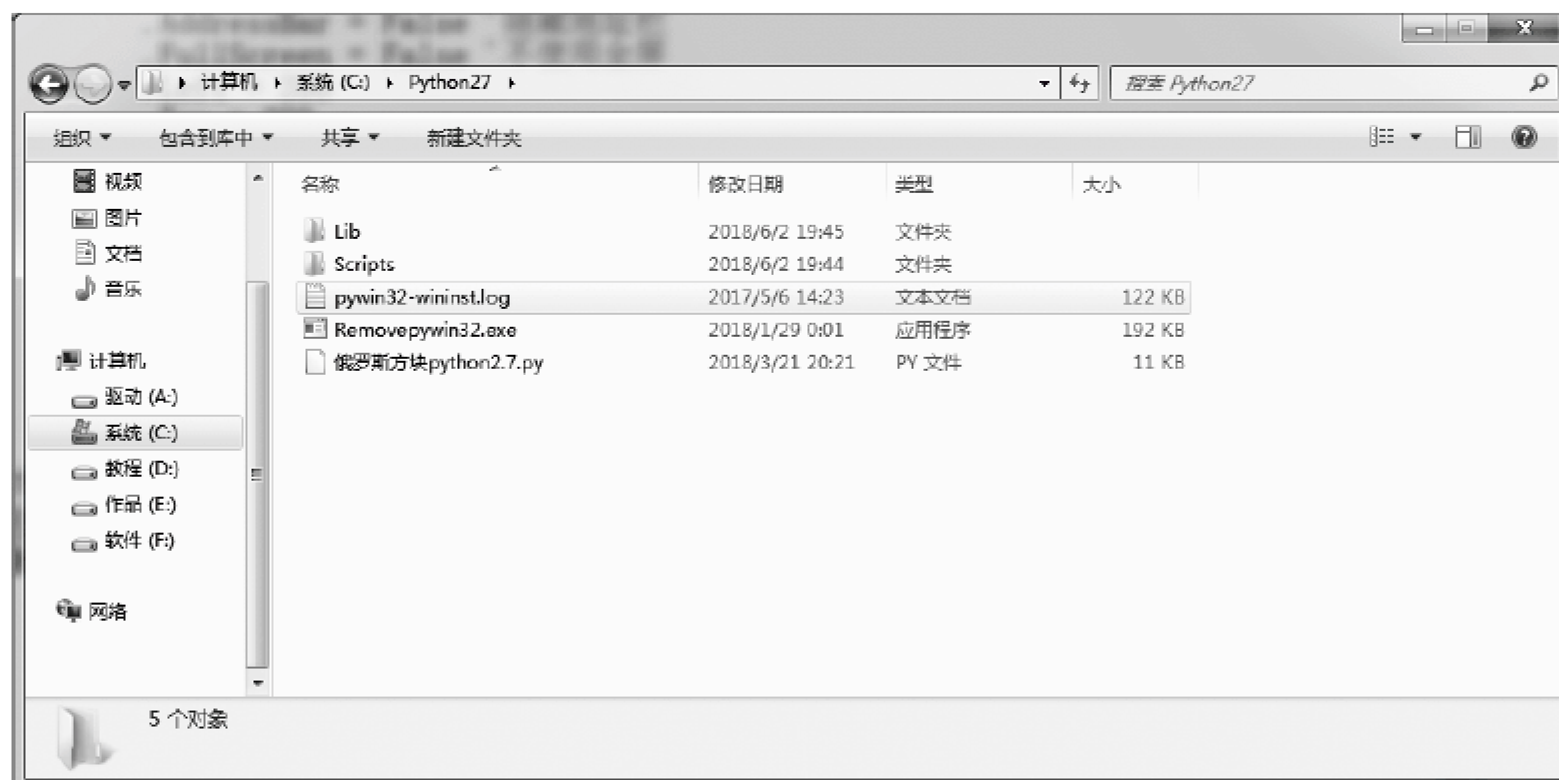


图 13-41 获取和操作资源管理器窗口

程序运行结束后，自动关闭上述窗口。

以上程序的源代码文件为“实例文档 89.xlsm”。

13.6 XMLHTTP

XMLHTTP 是一种浏览器对象，可用于模拟 HTTP 的 GET 和 POST 请求。XMLHTTP 提供客户端同 HTTP 服务器通信的协议。客户端可以通过 XMLHTTP 对象向 HTTP 服务器发送请求并使用微软 XML 文档对象模型处理回应。

XMLHTTP 用于网页自动化方面，可以打开一个指定的网址，并且发送数据到服务器，服务器会返回相应的信息，然后从返回的信息中提取关注的、有用的信息。因此，这个步骤也可以称为“网页数据抓取”“网抓”。

XMLHTTP 与 Internet Explorer、WebBrowser 有很大的不同，主要体现在以下几个方面。

- XMLHTTP 没有任何界面、浏览器。
- XMLHTTP 的特点是“发送 - 接收”，或者是“请求（Request）- 响应（Response）”，只重视发出去了什么，得到了什么，并不关注网页中是如何变化的，网页中有哪些元素。不像 Internet Explorer 那样，整个网页操作流程都要走一遍。
- 在网页解析方面，XMLHTTP 只返回网页源代码（一个很长的字符串），如果要用 HTML DOM 解析，还需要把得到的网页源代码赋给 HTMLDocument 对象。
- XMLHTTP 不能对网页元素进行操作。

使用 XMLHTTP 的作用和意义，暂时可以简单理解为根据指定的 url 返回网页源代码。

13.6.1 使用 XMLHTTP 的基本流程

对于访问不同的网站，XMLHTTP 的代码写法有所不同，但大致流程如下。

（1）为工程添加“Microsoft XML, v 6.0”的引用。

（2）创建 XMLHTTP 对象。

（3）使用 Open 方法打开指定的 url。

（4）使用 SetRequestHeader 设置请求头（不是必需）。

（5）使用 Send 方法发送请求。

（6）根据 ReadyState、Status 属性设置延时等待。

（7）对服务器返回的响应消息（ResponseBody、ResponseText 等）进行分析处理。

下面的程序根据上述流程，使用 XMLHTTP 打开一个博文的网址，获取网页源代码。

```
Public X As MSXML2.XMLHTTP60
Public H As MSHTML.HTMLDocument
Public SourceCode As String
Sub GET 请求 ()
    Set X = New XMLHTTP60
    With X
```

```
        .Open bstrMethod:="GET", bstrUrl:="https://www.cnblogs.com/ryueifu-VBA/p
/9128570.html", varAsync:=False
        .send
        Do Until .readyState = 4 And .Status = 200
            DoEvents
        Loop
        SourceCode = .responseText
        Debug.Print SourceCode
    End With
End Sub
```

代码分析：XMLHTTP 对象的 send 方法需要一定的时间才能把请求发送到服务器，因此要在调用 Send 方法之后加入循环体进行延时等待，而不能在 Send 方法之后立即获取其响应消息。

运行上述程序，立即窗口打印出该网页的 HTML 源代码，如图 13-42 所示。

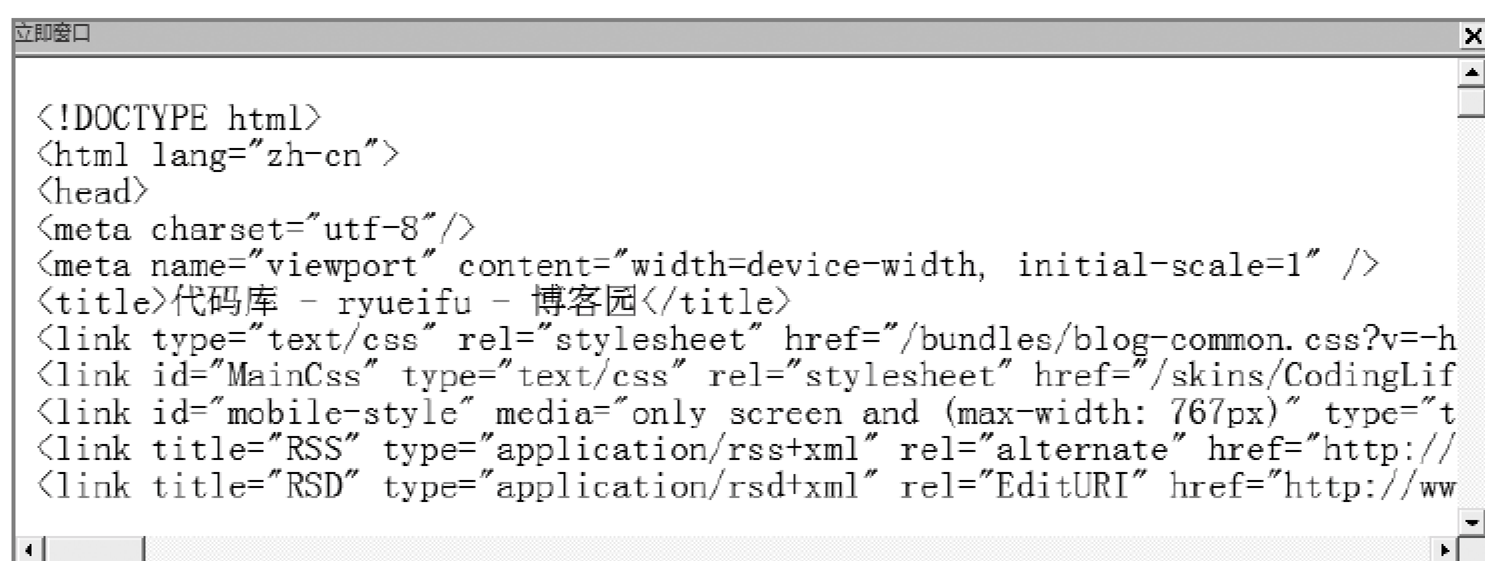


图 13-42　打印网页源代码

得到网页的源代码可以用在什么地方，这要根据具体的业务需求进行进一步的解析和利用。

以上过程是使用 XMLHTTP 进行网页请求和响应的典型模板，接下来讲解各个细节。

13.6.2　判断是否联网

访问外部网页的前提是计算机联网，计算机是否联网也可以用代码进行判断。

```
Public Declare Function InternetGetConnectedState Lib "wininet.dll" (ByRef
lpdwFlags As Long, ByVal dwReserved As Long) As Long
Sub 判断是否联网 ()
    If InternetGetConnectedState(&H0, &H0) = 1 Then
        MsgBox "已经联网", vbInformation
    Else
        MsgBox "断网状态", vbInformation
    End If
End Sub
```

运行上述程序，如果计算机已经联网，则弹出“已经联网”的提示对话框。

13.6.3　GET 和 POST 请求

GET 是向服务器发索取数据的一种请求，而 POST 是向服务器提交数据的一种请求，要提交的数据位于信息头后面的实体中。两者都是向服务器请求并且获得响应，使用 GET

作为请求方法时，XMLHTTP 对象的 Open 方法中只需要指定请求的 url，Send 方法不需要发送任何数据。使用 POST 请求时，XMLHTTP 对象的 Open 方法中也需要指定请求的 url，Send 方法后面添加要发送的数据。

假设 X 是一个 XMLHTTP 对象，使用 GET 请求时的基本语法格式如下。

```
With X
    .Open bstrMethod:="GET", bstrUrl:=" 网址 ", varAsync:=False
    .send
End With
```

使用 POST 请求时的语法格式如下。

```
With X
    .Open bstrMethod:="POST", bstrUrl:=" 网址 ", varAsync:=False
    .send postdata
End With
```

其中，postdata 是提交到服务器的数据信息。

在实际开发过程中，对于一个给定的 url，到底用 GET 还是用 POST，一种方法是根据经验判断，另一种方法是借助浏览器的开发工具来分析请求过程。

一般情况下，上网的时候在浏览器的地址栏中直接输入网址并按回车键，网页呈现相应的内容，这个操作其实就是一个 GET 请求，因为除了输入网址以外，没有提交其他的数据信息。

反过来，如果在网页中填写了一些信息，例如网站、论坛的登录实际上就是把用户名和密码发送到服务器，这个操作就是一个 POST 请求。

更加可靠的分析方法是使用浏览器的开发工具或者 Fiddler 来实时监视网络传输。

例如，在一个根据地名查询邮政编码的网站，按下 F12 键打开开发工具，切换到“网络”选项卡，并且把左侧的录制按钮设置为红色方块，如图 13-43 所示。



图 13-43　开发工具的“网络”选项卡

在地名文本框中输入“苏州”，单击网页中的“查询”按钮，等到页面刷新完毕后，在

开发工具窗格看到很多记录，这些记录显示在一个列表中，默认处于“摘要”视图。

从“摘要”视图中可以了解到每条请求记录的 url、请求方法、结果等信息，如图 13-44 所示。



图 13-44　每条请求记录的摘要

选中一条记录，然后切换至“详细信息”选项卡。可以看到该条记录的请求方法是 GET，请求 url 是 http://www.ip138.com/post/search.asp?area=%CB%D5%D6%DD&action=area2zip，如图 13-45 所示。

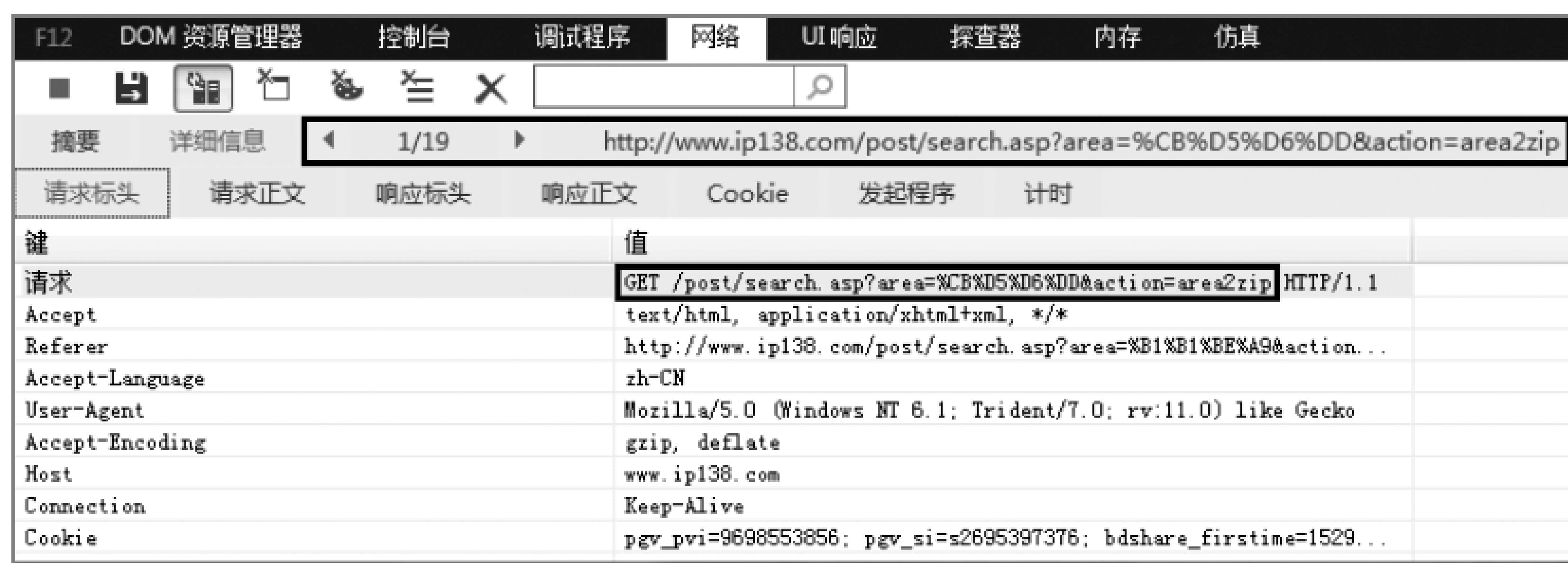


图 13-45　分析请求 url

根据直觉判断，url 中的 %CB%D5%D6%DD 是“苏州”的编码。

经过以上分析，如果要查询不同地区的邮政编码，只需要在 url 中替换为城市名称的编码即可。

下面的程序自动查询多个城市的邮政编码。

```
Sub 邮政编码查询()
    Dim city
     Const TemplateURL As String = "http://www.ip138.com/post/search.asp?area=
#1&action=area2zip"
    Set X = New XMLHTTP60
    For Each city In Array("重庆", "呼和浩特", "昆明")
        With X
```

```
                .Open bstrMethod:="Get", bstrUrl:=Replace(TemplateURL, "#1", URLEncode
(CStr(city))), varAsync:=False
                .send
                Do Until .readyState = 4 And .Status = 200
                    DoEvents
                Loop
                SourceCode = VBA.StrConv(.responseBody, vbUnicode, &H804)
                Debug.Print city, SourceCode
            End With
        Next city
    End Sub
```

代码分析：上述程序中常量 TemplateURL 是一个模板网址，只需要把具体城市名称的编码结果替换模板网址中的 #1 占位符即可作为 XMLHTTP 的 url。

由于该网站的编码是 GB2312，因此获取源代码时，不能直接用 ResponseText，而是用 StrConv 转换一下。

运行上述程序，在立即窗口中打印出每次查询的网页源代码，然后利用字符串处理方法取出邮政编码即可，如图 13-46 所示。



图 13-46　批量查询不同城市的邮政编码

以上实例是 XMLHTTP 对象的 GET 请求的典型范例。POST 请求方面的应用在 13.7 节中进行讲解。

13.6.4　正确获取网页源代码

根据指定的网址获取对应的网页源代码，对网页自动化、网页数据获取的意义不言而喻。在 XMLHTTP 的使用过程中，当执行 Send 方法后，加入必要的延时处理，就可以获取网页源代码。

XMLHTTP 对象提供了 ResponseBody、ResponseText 等属性。

ResponseBody 是一个没有经过任何转换加工的二进制数据包，其中包含了服务器返回的所有内容，是一个字节数组。

ResponseText 是一个字符串。两者可以进行相互转换。

其中，当网页的编码是 UTF-8 或 GBK 时，使用 ResponseText 属性可以得到正常显示的网页源代码，当网页编码是 GB2312 时，使用 ResponseText 得到的网页源代码有大量的乱码。出现乱码的情况下，不能直接使用 ResponseText 属性，需要把 ResponseBody 按照网页编码转换为字符串。

在获取源代码之前，可以使用 ADODB 的 Stream 对象判断一下 ResponseBody 是不是 UTF-8。

下面的这个自定义函数用来判断指定的二进制数组是不是 UTF-8 编码。

```
Public Function IsUTF8(b() As Byte) As Boolean
    Dim i As Long, AscN As Long, Length As Long
    Length = UBound(b) + 1
    If Length < 3 Then
        IsUTF8 = False
    Exit Function
    ElseIf b(0) = &HEF And b(1) = &HBB And b(2) = &HBF Then
        IsUTF8 = True
    Exit Function
    End If
    Do While i <= Length - 1
        If b(i) < 128 Then
            i = i + 1
            AscN = AscN + 1
        ElseIf (b(i) And &HE0) = &HC0 And (b(i + 1) And &HC0) = &H80 Then
            i = i + 2

        ElseIf i + 2 < Length Then
            If (b(i) And &HF0) = &HE0 And (b(i + 1) And &HC0) = &H80 And (b(i + 2)
And &HC0) = &H80 Then
                i = i + 3
            Else
                IsUTF8 = False
            Exit Function
            End If
        Else
            IsUTF8 = False
        Exit Function
        End If
    Loop
    If AscN = Length Then
        IsUTF8 = False
    Else
        IsUTF8 = True
    End If
End Function
```

在下面的程序中，无论用 XMLHTTP 访问哪一个网址，都可以返回正常显示的网页源代码。实现原理是首先为工程添加 ADODB 的外部引用，然后使用上述 IsUTF8 函数对 XMLHTTP 对象的 ResponseBody 进行编码判断，判断编码的目的是给 ADODB.Steam 对象指定编码。

```
Sub 获取网页源代码()
     Dim b() As Byte
     Dim objstream As ADODB.Stream
     Dim SourceCode As String
     Set X = New XMLHTTP60
     With X
        .Open "GET", "http://www.gdchess.com/", False
```

```
            .send
            Do Until .readyState = 4 And .Status = 200
                DoEvents
            Loop
            b = .responseBody '赋给字节数组
            Set objstream = New ADODB.Stream
            With objstream
                .Type = ADODB.adTypeBinary '二进制模式
                .Mode = ADODB.adModeReadWrite
                .Open
                .Write Buffer:=b
                .Position = 0
                .Type = adTypeText '文本模式
                If IsUTF8(b) Then
                    .Charset = "utf-8"
                Else
                    .Charset = "GB2312"
                End If
                SourceCode = .ReadText '返回文本
                Debug.Print SourceCode
                .Close
            End With
        End With
    End Sub
```

可以尝试各种网址，运行上述程序后，在立即窗口均能正确返回网页源代码。得到网页源代码后，可以使用 VBA 的字符串处理、正则表达式来提取其中有用的内容，也可以把网页源代码赋给 HTMLDocument 对象，使用 HTML DOM 进行分析。

13.6.5　网页中文件的下载

在网页自动化方面，经常需要从一个网站批量下载网页中超链接指向的文件，网页上的文件一般是一个超链接 a 元素。文件的实际 url 显示在 a 元素的 href 属性中。

XMLHTTP 对象的 Open 方法，除了可以打开一般的网址，也可以直接指定一个文件的地址。当用 XMLHTTP 请求文件时，返回的二进制数据 ResponseBody 就是这个文件本身，保存为本地文件即可实现文件下载。

例如，压缩文件 WinRAR 的主页有很多安装程序的下载地址，如果要下载该页面显示的所有文件，首先要获取该页面上所有的超链接。获取到超链接后，依次下载每一个文件。

事先使用浏览器的开发工具检查其中一个超链接，可以看到超链接的 href 是一个相对地址，如图 13-47 所示。

必须使用绝对地址才能下载，因此需要把该 href 与网站主页地址拼接为：https://www.rarlab.com/rar/wrar56b5tc.exe。

```
Public X As MSXML2.XMLHTTP60
Public H As MSHTML.HTMLDocument
Public SourceCode As String
Sub 解析网页内容()
    Dim FileLink As MSHTML.HTMLAnchorElement
```

```
        Set X = New XMLHTTP60
        With X
            .Open bstrMethod:="Get", bstrUrl:="https://www.rarlab.com/", varAsync:=
False
            .send
            Do Until .readyState = 4 And .Status = 200
                DoEvents
            Loop
            SourceCode = .responseText
        End With
        Set H = New MSHTML.HTMLDocument
        H.body.innerHTML = SourceCode                    '把网页源代码赋给 HTMLDocument 文档
        For Each FileLink In H.getElementsByTagName("a")
            If FileLink.href Like "*.exe" And FileLink.innerText = "32 bit" Then
                Debug.Print FileLink.href, FileLink.innerText
            End If
        Next FileLink
    End Sub
```

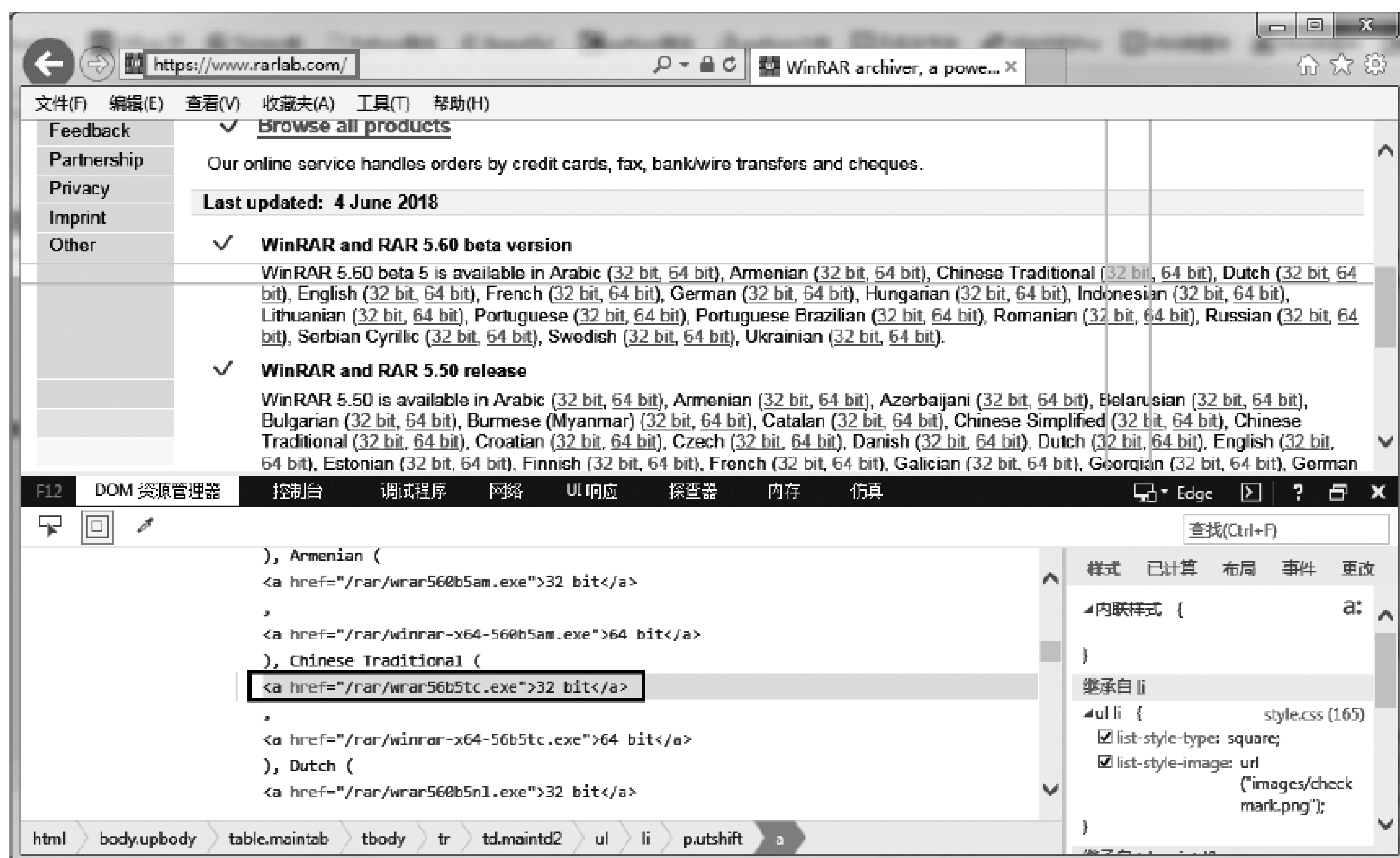


图 13-47 超链接中的相对路径

代码分析：由于主页上的超链接 a 元素均没有 id 属性，因此使用 getElementsByTagName("a") 获取页面上的所有超链接，为了避免遍历到广告或者其他没用的超链接，可以使用 If 语句进行过滤，只有以 .exe 结尾的超链接且文本内容是 32 bit 的才遍历到。

运行上述程序，在立即窗口打印出 32 位安装程序的超链接地址，如图 13-48 所示。

这里假定要下载第 3 个文件。下面的程序中，XMLHTTP 的 url 设置为文件的绝对路径，发送请求后，把 ResponseBody 保存为本地文件。

```
立即窗口
about:/rar/wrar560b5ar.exe   32 bit
about:/rar/wrar560b5am.exe   32 bit
about:/rar/wrar56b5tc.exe    32 bit
about:/rar/wrar560b5nl.exe   32 bit
about:/rar/wrar56b5.exe      32 bit
about:/rar/wrar560b5fr.exe   32 bit
about:/rar/wrar56b5d.exe     32 bit
about:/rar/wrar56b5hu.exe    32 bit
about:/rar/wrar560b5id.exe   32 bit
about:/rar/wrar560b5lt.exe   32 bit
about:/rar/wrar560b5pt.exe   32 bit
```

图 13-48 遍历网页中扩展名为 .exe 的文件地址

```
Sub 下载文件 ()
    DownLoadFile url:="https://www.rarlab.com/" & Replace("about:/rar/wrar56b5tc.exe", "about:/", ""), LocalName:="WinRAR 32 位中文版 .exe"
End Sub
Sub DownLoadFile(url As String, LocalName As String)
     Dim b() As Byte
     Set X = New XMLHTTP60
     With X
        .Open "GET", url, False
        .send
        Do Until .readyState = 4 And .Status = 200
            DoEvents
        Loop
        b = .responseBody
        Dim FileNum As Long
        FileNum = FreeFile
        Open ThisWorkbook.Path & "\" & LocalName For Binary Access Write As #FileNum
            Put #FileNum, , b
        Close #FileNum
    End With
End Sub
```

运行上述程序，在工作簿路径下多了一个文件，如图 13-49 所示。

名称	修改日期	类型	大小
Winrar 32位中文版.exe	2018/6/18 13:00	应用程序	3,140 KB
实例文档95.xlsm	2018/6/18 11:00	Microsoft Excel ...	21 KB

图 13-49　根据文件的 url 下载文件到本地计算机

以上程序的源代码文件为“实例文档 95.xlsm”。

13.6.6　使用 API 函数下载文件

如果知道网络文件的 url，也可以使用 API 函数下载文件到本地计算机。

```
Public Declare Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" (ByVal pCaller As Long, ByVal szURL As String, ByVal szFileName As String, ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long
Public Sub 下载文件 ()
    Dim r As Long
     r = URLDownloadToFile(0, "https://files.cnblogs.com/files/ryueifu-VBA/%E4%BB%A3%E7%A0%81%E5%BA%93.rar", ThisWorkbook.path & "\Temp.rar", 0, 0)
    If r = 0 Then
        MsgBox " 下载成功！ "
    Else
        MsgBox " 下载失败！ "
    End If
End Sub
```

运行上述程序，在工作簿路径下产生一个 Temp.rar 压缩包文件。

以上程序的源代码文件为“实例文档 96.xlsm”。

13.7 WinHttp

Microsoft Windows HTTP 服务（WinHttp）为开发人员提供了 HTTP 客户端应用程序编程接口（API），通过 HTTP 协议向其他 HTTP 服务器发送请求。

使用 WinHttp 也可以实现网页数据的发送和获取，WinHttp 对象的属性、方法与 XMLHTTP 大部分相同，本节介绍使用 WinHttp 向网站服务器发送数据、获取服务器返回数据的方法。

平时上网的过程中，经常需要从网站上进行查询、转换、计算，或者提交注册信息、发表博客、论坛账号登录，而不是简简单单通过 url 打开一个网页查看内容。尤其是访问一些需要登录的网站，登录前后能够访问的范围有很明显的差别。

本节使用 WinHttp 自动向网站提交账户信息，登录成功后进一步访问网站中的其他页面的信息。

13.7.1 POST 请求和响应

VBA 中使用 WinHttp 对象，首先需要添加对“Microsoft WinHTTP Services, version 5.1”的外部引用，如图 13-50 所示。

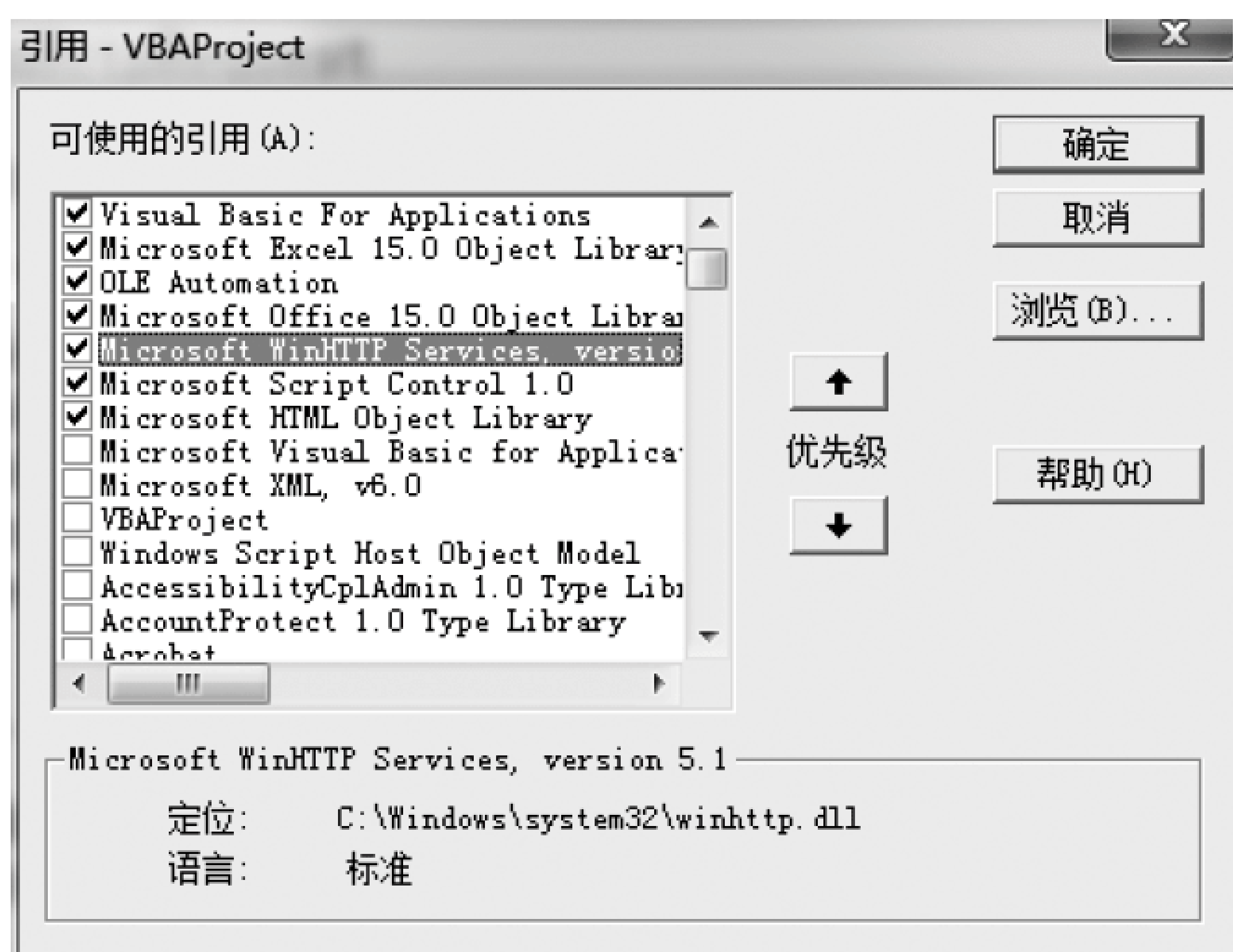


图 13-50 添加外部引用

发送 POST 请求的过程，就是向网站提交数据的过程，可以分为请求（Request）和响应（Response）两大部分，请求和响应往往是有对应关系的，根据提交的信息，服务器做出相应的回答。

请求部分需要提供如下三部分内容。

- ❑ 请求 url：POST 请求的网址。
- ❑ 请求头（RequestHeader）：是请求报文特有的，它们为服务器提供了一些额外信息，例如客户端希望接收什么类型的数据。

- 请求正文（RequestBody）：向服务器发送的数据，如果是登录网站，用户名和密码往往要包含在请求正文中。

请求发送后，服务器返回的响应消息部分如下。

- 响应头（ResponseHeader）：响应头向客户端提供一些额外信息，例如谁在发送响应、响应者的功能，甚至与响应相关的一些特殊指令。这些头部有助于客户端处理响应，并在将来发起更好的请求。
- 响应正文（ResponseBody）：是服务器返回的资源的内容。

在使用 WinHttp 发送 POST 请求时，请求部分的代码编写，需要事先手工在网页上操作，利用浏览器开发工具的“网络”选项卡，录制请求和响应过程。

13.7.2　抓包分析

在手工操作网页的过程中，按 F2 键打开浏览器的开发工具 F12，切换至“网络”选项卡，并且把左侧的“开始录制”按钮切换为红色方块。当网页上发生请求行为时，开发工具窗格会自动记录每次请求的详细信息。

此处以登录网站并且查看个人资料为例，逐步讲解 WinHttp 的实现过程。

首先在浏览器中打开账户登录的网页，输入用户名和密码之后，再打开开发工具，并设置为录制状态，如图 13-51 所示。



图 13-51　录制请求过程

然后单击“登录”按钮，成功登录网站，并且在开发工具窗格看到多了很多条请求记录，定位到方法为“POST”的那条记录，如图 13-52 所示。

然后从“摘要”页切换至“详细信息”页，或者直接双击请求记录，打开该条请求的详细页面。“详细信息”中又分为“请求标头”“请求正文”“响应标头”“响应正文”等选项卡。



图 13-52 找到对应于登录过程的那条请求

在"请求标头"中，可以获取到 POST 请求的 url、请求标头。比较重要的请求头有：Referer、Content-Type、Cookie 等，如图 13-53 所示。

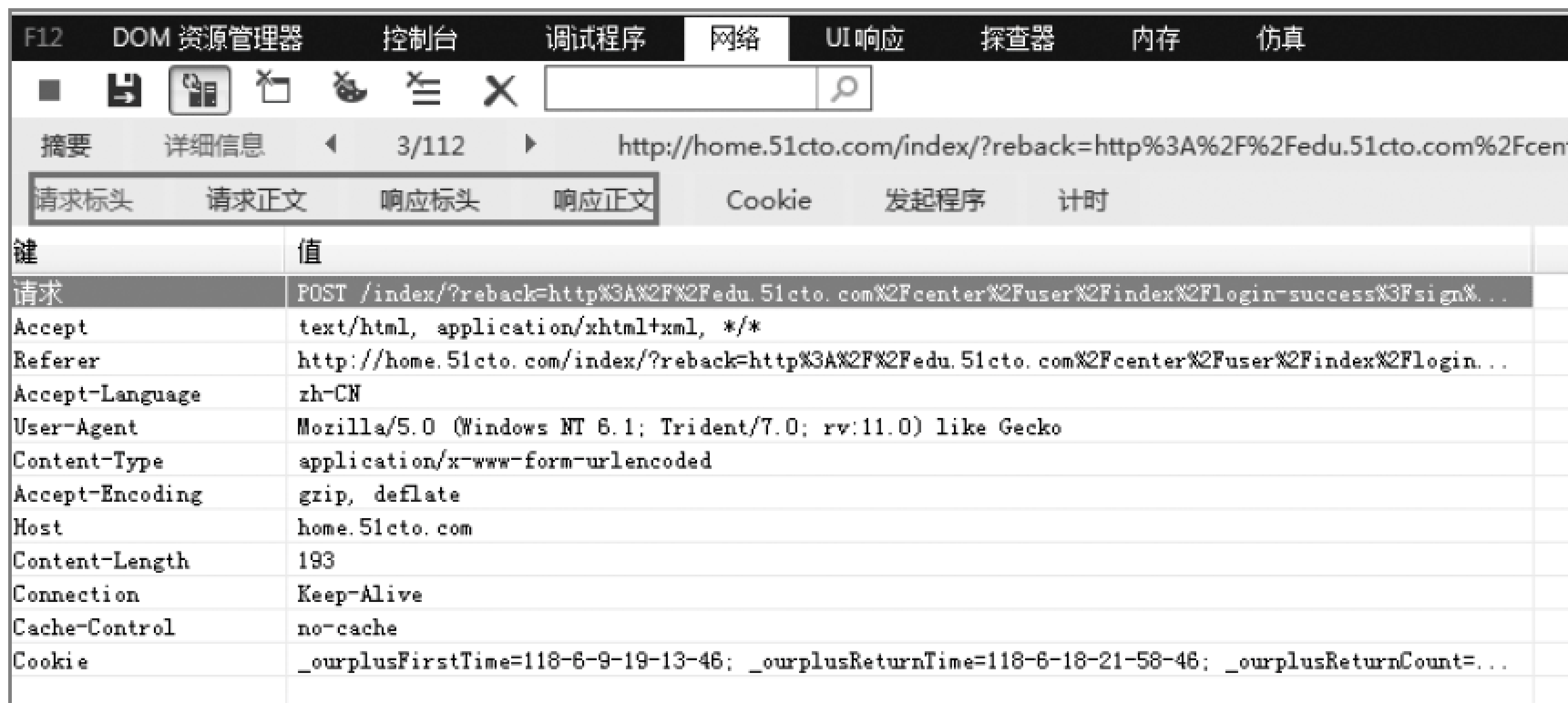


图 13-53 一条请求的详细信息

"请求正文"中，可以看到登录网站过程中提交到网站服务器的数据，这些数据以"键值对"的形式提交给服务器，键和值之间用 = 连接，每个键值对之间用 & 隔开。

具体书写形式类似于：user=ryueifu&password=123456&other=...，如果请求正文中包含中文或其他符号，还需要进行编码，如图 13-54 所示。



图 13-54 请求正文

一般来说，获取到请求 url、请求标头、请求正文这 3 项，就可以动手写代码了。

顺便了解一下“响应标头”，实际上无论是手工在页面中操作，还是用程序自动操作，返回的“响应标头”应该是一致的。

响应标头也是一些键值对，在开发工具中显示为表格形式，如图 13-55 所示。

F12　DOM 资源管理器　控制台　调试程序　网络　UI 响应　探查器　内存　仿真

摘要　详细信息　3/113　http://home.51cto.com/index/?reback=http%3A%2F%2Fed

请求标头　请求正文　响应标头　响应正文　Cookie　发起程序　计时

键	值
响应	HTTP/1.1 200 OK
Date	Mon, 18 Jun 2018 13:58:59 GMT
Content-Type	text/html; charset=UTF-8
Transfer-Encoding	chunked
Connection	keep-alive
Server	nginx
P3P	CP="CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE ...
Expires	Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control	no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma	no-cache
Set-Cookie	PHPSESSID=2ntc4tfu317s6qs0oaktml80c7; path=/; HttpOnly
Set-Cookie	pub_sid=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-A...
Set-Cookie	pub_sauth1=Ex1MAQwDQmsFAgUCVApVPQUGB11VAQUCUVo; path=/; domai...
Set-Cookie	pub_sauth2=60417421d791dbe44bfcc315fe1bbf9a; path=/; domain=....
Set-Cookie	pub_cookietime=0; path=/; domain=.51cto.com; httponly
refresh	0;url=http://edu.51cto.com/center/user/index/login-success?si...

图 13-55　响应标头

通过响应标头往往可以看出请求的结果、登录是否成功等信息。

注意　其他种类的浏览器，开发工具的外观设计有所不同，但是录制出的内容基本一样。

13.7.3　构建代码

利用浏览器的开发工具的抓包分析得到的数据，内容往往比较长，不太适合直接写在 VBA 代码中，实际开发过程中，可以把这些参数存储于文本文件或 Excel 单元格中。

例如单元格 B2 存储 POST 请求的 URL，B3-B5 存储请求标头，B6 存储请求正文，如图 13-56 所示。

实例文档98.xlsm - Excel

文件　开始　插入　页面布局　公式　数据　审阅　视图　开发工具　加载项

E15

	A	B	C
1	Param	Value	
2	Post URL	http://home.51cto.com/index/?reback=http%3A%2F%2Fedu.51cto.com%2Fcenter%2Fuser%2Findex%2Flogin-su	
3	Referer	http://home.51cto.com/index/?reback=http%3A%2F%2Fedu.51cto.com%2Fcenter%2Fuser%2Findex%2Flogin-su	
4	Content-Type	application/x-www-form-urlencoded	
5	Cookie	_ourplusFirstTime=118-6-9-19-13-46; _ourplusReturnTime=118-6-18-21-58-46; _ourplusReturnCount=6;	
6	Post Data	_csrf=VGxEQWVFbjNRXHcTNyvFBh8IEzAgCDFHAh8GDgR2GWpkFAV1EzE%2FWQ%3D%3D&LoginForm%5Busername%5D=ryue	
7			

图 13-56　使用单元格存储参数

为了能让 WinHttp 对象在账号登录之后，在其他过程中继续访问网站中的其他网页，要把该对象声明为 Public。

下面的程序自动登录网站，并且在立即窗口打印响应标头信息。

```
Public W As WinHttp.WinHttpRequest
```

```
Sub 账号登录()
    Set W = New WinHttp.WinHttpRequest
    With W
        .Open Method:="POST", URL:=Range("B2").Value, Async:=False
        .SetRequestHeader "Referer", Range("B3").Value
        .SetRequestHeader "Content-Type", Range("B4").Value
        .SetRequestHeader "Cookie", Range("B5").Value
        .Send body:=Range("B6").Value
        Debug.Print .GetAllResponseHeaders
        Debug.Print .GetResponseHeader(Header:="Content-Type")
    End With
End Sub
```

运行上述程序，在立即窗口打印出了所有响应标头，如图 13-57 所示。

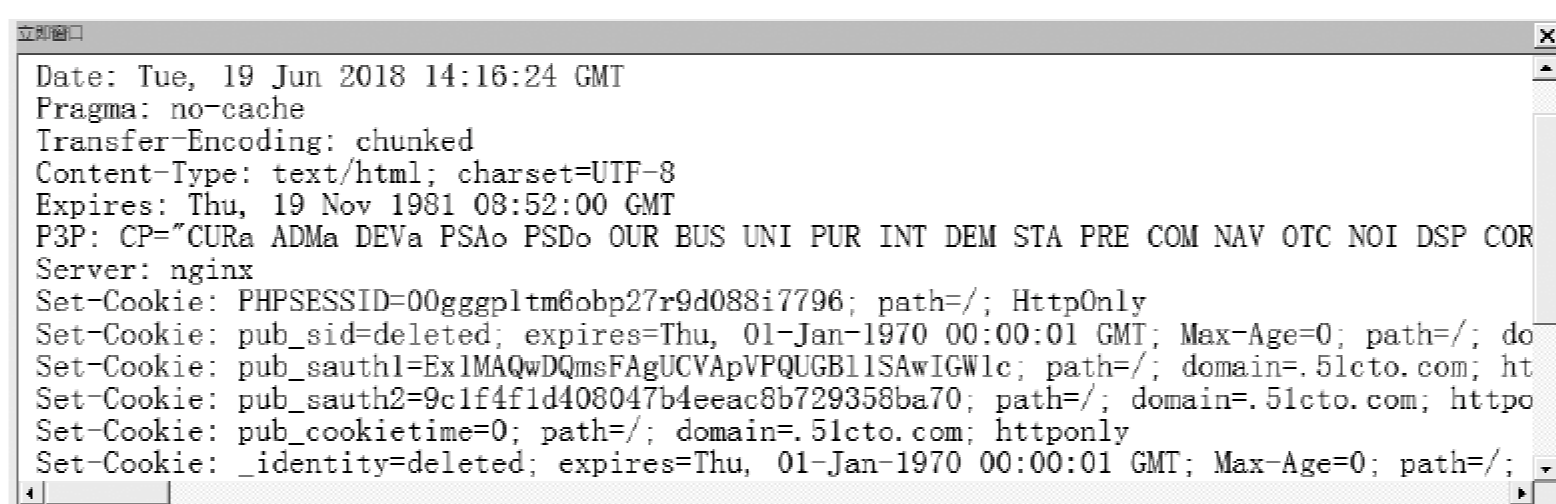


图 13-57　自动发送请求并打印响应标头

将这些响应标头与抓包分析得到的响应标头对比，可以发现两者基本一致，说明登录成功。

那么提交错误的用户名或密码，结果会怎样呢？假设故意写错用户名或密码，再次运行上述程序，会发现少了很多响应标头，尤其缺少了 Set-Cookie 响应标头，如图 13-58 所示。

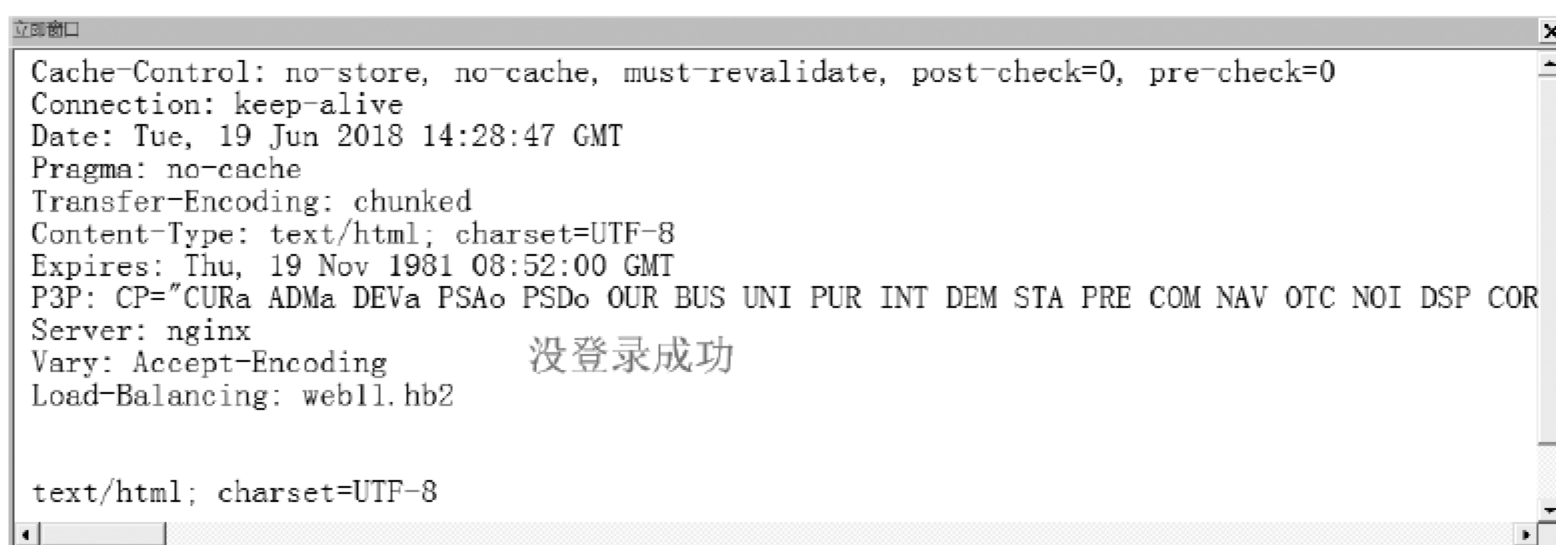


图 13-58　使用错误的账号信息所返回的响应标头

13.7.4　继续访问网站其他网页

由于声明的 WinHttp 是一个模块级公有变量，当登录过程结束后，该变量依然保留着登录状态的信息，因此可以使用它继续访问同一网站的其他网页。

```
Sub 访问个人主页()
    Dim H As MSHTML.HTMLDocument
    Dim MyInformation As MSHTML.HTMLDivElement
    With W
        .Open Method:="GET", URL:="http://edu.51cto.com/center/course/lecturer/
```

```
course", Async:=False
            .Send
            Set H = New MSHTML.HTMLDocument
            H.body.innerHTML = .ResponseText
            Set MyInformation = H.getElementsByClassName("user_Top").Item(0)
            Debug.Print MyInformation.innerText
        End With
    End Sub
```

运行上述过程，立即窗口打印出讲师的个人信息，如图 13-59 所示。

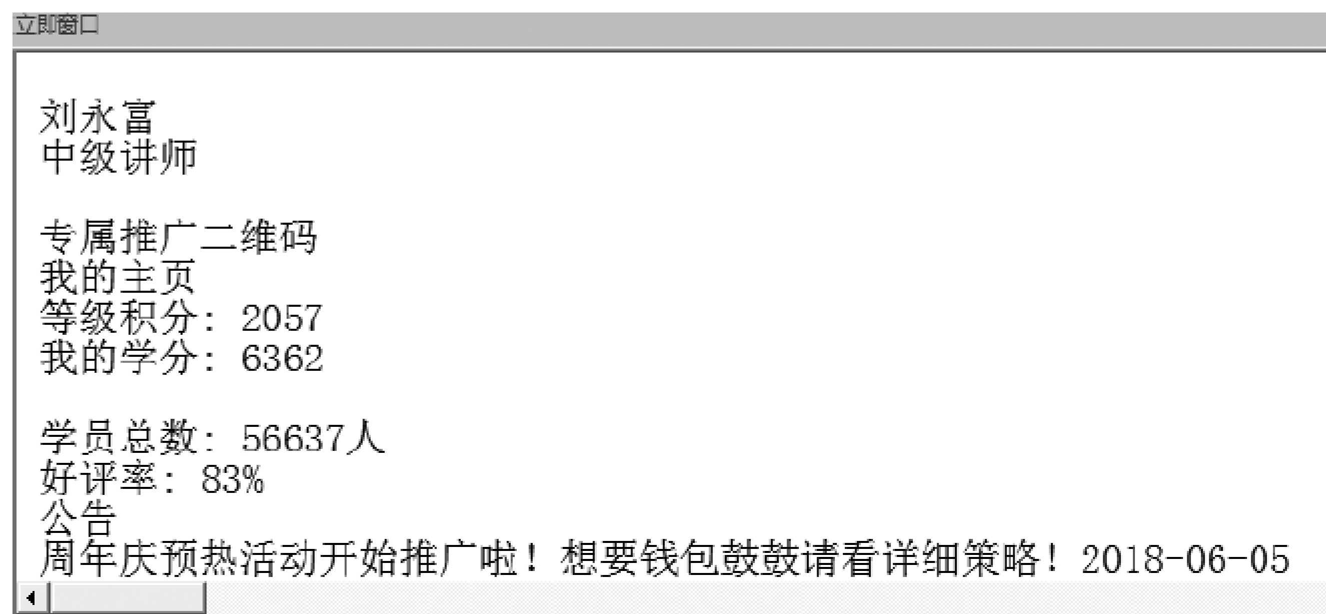


图 13-59　在登录状态下访问其他网页

以上程序的源代码文件为“实例文档 97.xlsm”。

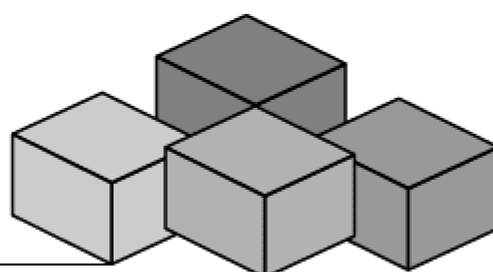
13.8　本章小结

HTMLDOcument 可以把一个字符串形成 HTML 对象模型，从而对网页中的元素进行获取、操作、编辑等操作。

Internet Explorer 和 WebBrowser 是实际的浏览器对象，使用 VBA 代码可以代替鼠标和键盘，自动操作网页。

XMLHTTP、WinHttp 二者和浏览器、网页无关，这两个对象需要的原料是网址、请求标头、发送的数据，执行 Send 方法以后，返回的是响应标头、ResponseBody 等。

第 14 章
其他常见话题



编程开发过程中，经常会用到随机数、进制转换、颜色的获取和设置、日期和时间的表达及转换等知识。

14.1　随机数

VBA 中使用 Randomize 语句初始化随机数生成器种子，Rnd() 可以产生一个介于 0 和 1 之间的随机小数。

```
Sub Test1()
    Dim number As Double
    Randomize                    '初始化随机数生成器来产生种子
    number = Rnd()               ' 0 到 1 之间的小数
    Debug.Print number
End Sub
```

多次运行上述过程，每次打印的结果均不一样。

由于 0 ≤ Rnd() ≤ 1，假设有一个正整数 n，可以推出：

```
0 ≤ n*Rnd() ≤ n
```

进一步推出：CInt(n*Rnd()) 是一个介于 0 和 n 之间的随机整数。

例如，3 + CInt((7 － 3) * Rnd()) 会产生 3 到 7 之间的随机整数。

14.2　进制

VBA 编程中支持八进制和十六进制，八进制常量用 &O 作为前缀，十六进制常量用 &H 作为前缀。

Oct 函数用于把其他数字转换为八进制字符串，Hex 函数用于把其他数字转换为十六进制字符串。

```
Sub 八进制和十六进制 ()
    Dim b As Integer, c As Integer
    b = &O23
    c = &H31
    Debug.Print b, c                          '自动转换为十进制数
    Debug.Print Oct(b), Hex(c)                '显示为八进制、十六进制字符串
End Sub
```

运行上述程序，打印结果如下。

```
19            49
23            31
```

其中,19 是八进制数 23 转换为十进制的结果,49 是十六进制数 31 转换为十进制的结果。因此，&O23 与 19 是等价的，&H31 与 49 是等价的，&HFF 与 255 也是等价的。

14.3　颜色

在编程过程中，经常会获取或设置对象的颜色，例如设置单元格的填充色。在 Excel VBA 中可以通过设置 Color 或 ColorIndex 属性来设置颜色。

Color 属性的取值可以是 VBA 中颜色常数之一，也可以使用 RGB 函数来设置。例如下面两行代码的作用相同，都是把单元格填充色变为红色。

```
Range("A1").Interior.Color = VBA.ColorConstants.vbRed
Range("B1").Interior.Color = RGB(255, 0, 0)
```

也就是说，vbRed 和 RGB(255, 0, 0) 以及 RGB(&HFF, 0, 0) 是完全相等的。

实际上，颜色是用三原色来描述的，也就是 R、G、B 混合起来的，每个分量的最小值是 0，最大值是 &HFF（255）。

假设有一种颜色的 R、G、B 的比例是 5:3:2，那么换成十进制是 255:153:102，表示为十六进制是 &HFF、&H99、&H66，以下两行代码均可在单元格中看到这种颜色的实际效果。

```
Range("A1").Interior.Color = &H6699FF
Range("B1").Interior.Color = RGB(&HFF, &H99, &H66)
```

另外，Excel VBA 还可以使用 ColorIndex 来表示颜色，ColorIndex 从 1 到 56 代表 56 种不同的颜色。

为了查看所有 ColorIndex 的效果，可以用如下的循环把每一个单元格填充不同的颜色。

```
Sub Test3()
    Dim i As Integer
    For i = 1 To 56
        Range("A" & i).Interior.ColorIndex = i
    Next i
End Sub
```

以上程序的源代码文件为“实例文档 99.xlsm”。

14.4 Excel 的文件格式

Excel 2007 以下文件（低版本文件）的扩展名是 .xls，这种格式的工作表最后一行的行号是 65536（256^2），最右一列的列标字母是 IV（第 256 列）。因此工作表最右下角的单元格地址是 IV65536。

Excel 2007 以上文件（高版本文件）的扩展名是 4 位英文字母，例如 .xlsx、.xlsm 等。这种文件的工作表最后一行的行号是 1048576（1024^2），最右一列的列标字母是 XFD（128^2），最右下角的单元格地址是 XFD1048576。

但是，文件的格式和 Excel 的版本又是两回事，即使 Excel 2013 中也支持低版本的 Excel 文件，也就是说在 Excel 2013 中打开扩展名为 .xls 的工作簿，最右下角的单元格依然是 IV65536。编程过程中，把数组赋给单元格的时候，或者使用 Range.CopyFromRecordset 方法把结果记录集粘贴到单元格，必须考虑单元格区域能否容纳这些数据。

下面的程序用来判断当前工作簿的文件类型。

```
Sub 判断工作簿类型 ()
    Dim fmt As Excel.XlFileFormat
    fmt = ActiveWorkbook.FileFormat
    If fmt = Excel.XlFileFormat.xlExcel8 Then
        Debug.Print " 当前工作簿是 Excel2003 文件 "
    ElseIf fmt = Excel.XlFileFormat.xlOpenXMLWorkbook Then
        Debug.Print " 当前工作簿是 .xlsx 文件 "
    ElseIf fmt = Excel.XlFileFormat.xlOpenXMLWorkbookMacroEnabled Then
        Debug.Print " 当前工作簿是 .xlsm 文件 "
    End If
End Sub
```

众所周知，在 Excel 中按下快捷键【Ctrl+N】快速新建一个工作簿 Book1，那么这个新建的而且尚未保存的工作簿是什么格式？这取决于 Excel 应用程序对默认文件格式的设置，如图 14-1 所示。

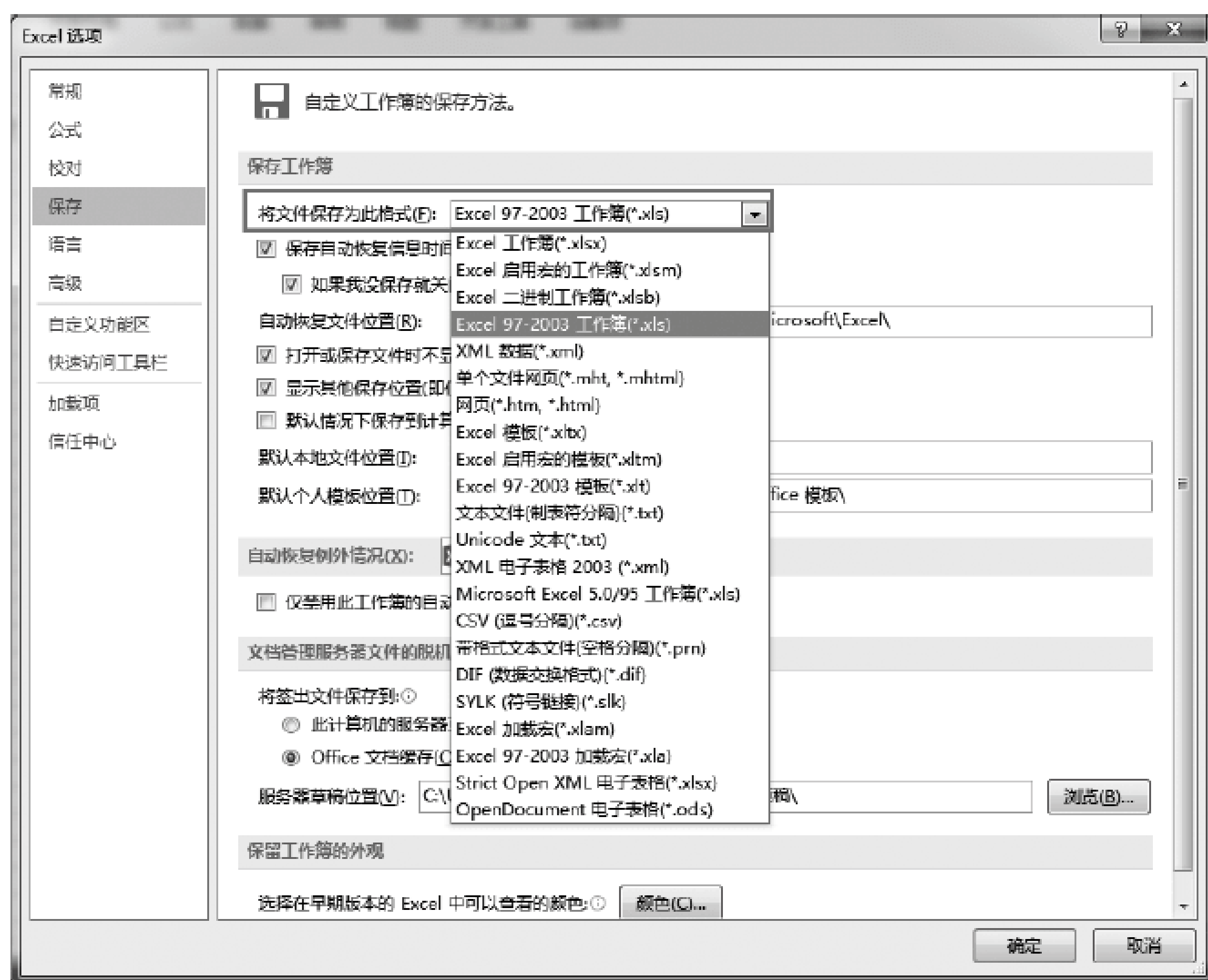


图 14-1 “Excel 选项”对话框

如果默认文件格式设置为 .xls 格式，那么新建的工作簿就是低版本文件。

Application.DefaultSaveFormat 用来读写 Excel 默认的文件格式。

下面的程序，首先把 Excel 默认文件格式设置为低版本文件格式，然后新建一个工作簿，并且试图把一个二维数组赋给单元格区域。

```
Sub 创建低版本的 Excel 文件 ()
    Dim wbk As Excel.Workbook, wst As Excel.Worksheet
    Dim arr(1 To 70000,1 To 300) As Integer
    Application.DefaultSaveFormat = Excel.XlFileFormat.xlExcel8
    Set wbk = Application.Workbooks.Add
    Set wst = wbk.Worksheets(1)
    wst.Range("A1").Resize(70000, 300).Value = arr
End Sub
```

运行上述程序，出现运行时错误，如图 14-2 所示。

出错的原因是：低版本文件不存在七万行、一千列。

如果把 Application.DefaultSaveFormat 修改为 Excel.XlFileFormat.xlOpenXMLWorkbook，再次运行上述程序，不会出错。

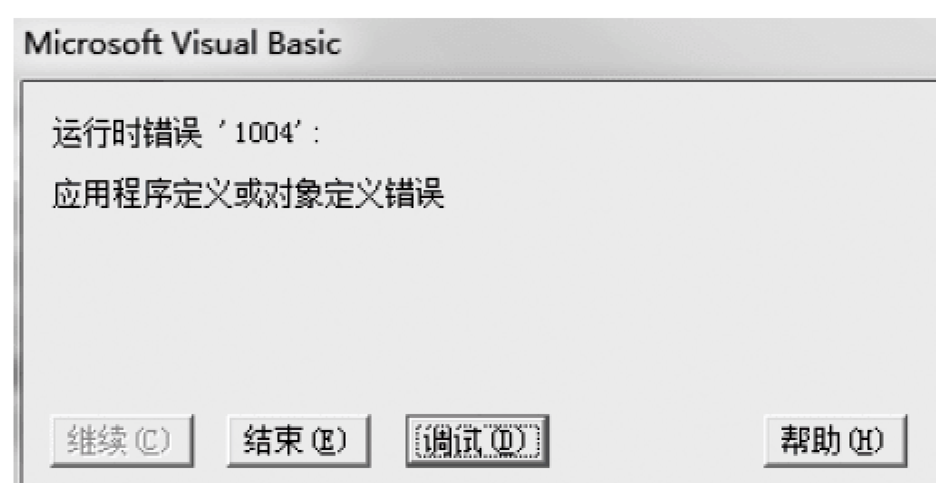


图 14-2　行列超出工作表范围

14.5　日期和时间运算

在 VBA 中，日期和时间是一种介于数字和字符串之间的数据类型，在实际编程过程中，遇到日期时间处理的场合非常多。

日期时间的数据类型是 Date，日期时间常量两边用 # 括起来。

内置函数 Now、Date、Time 分别返回当前日期和时间、当前日期、当前时间。

14.5.1　分量的提取

VBA 的内置函数 Year、Month、Day、Hour、Minute、Second 用于提取每个分量，均返回一个整数。

下面的程序打印一个日期的年月日、时分秒。

```
Sub 分量的提取 ()
    Dim dt As Date
    dt = #8/8/2008 9:15:37 AM#
    Debug.Print Year(dt), Month(dt), Day(dt), Hour(dt), Minute(dt), Second(dt)
End Sub
```

14.5.2　日期和时间的生成

用于生成日期、时间的 VBA 函数如下。

- ❑ DateSerial：由年月日三个分量合成日期。
- ❑ TimeSerial：由时分秒三个分量合成时间。
- ❑ CDate：可以把日期时间字符串、整数转换为日期时间。

```
Sub 日期和时间的生成()
    Debug.Print DateSerial(2015, 7, 8)
    Debug.Print TimeSerial(20, 17, 8)
    Debug.Print CDate("2015, 7, 8"), CDate("2015年7月8日 15时30分")
    Debug.Print CDate(20150)
End Sub
```

代码分析：CDate是一个功能强大的转换函数，当参数是一个字符串，而且是一个能够识别的日期，则可以成功转换。当参数是一个数字，则数字0相当于#1899/12/30 00:00:00#，例如：CDate(20.75)转换成日期#1900/1/19 18:00:00#。

CDbl(#1900/1/19 18:00:00 #)把日期时间转换为数字，结果是20.75。

运行上述过程，立即窗口的结果如图14-3所示。

> **注意** 字符串能否转换为日期时间，可以用IsDate判断，如果返回True，则可以用CDate进行转换。

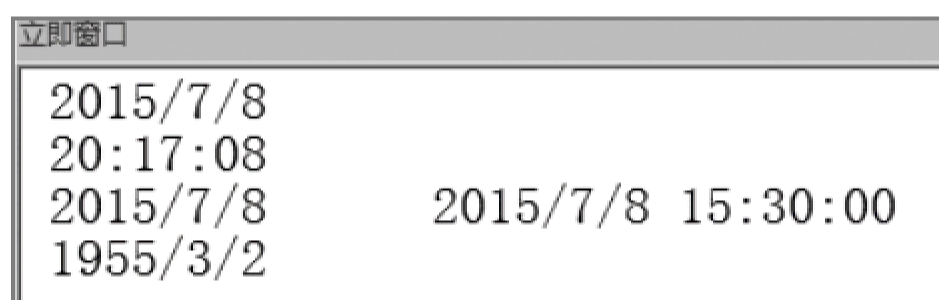


图 14-3 日期和时间的生成

```
Sub 事先判断()
    Dim s As String
    s = "15点30分21秒"
    If IsDate(s) Then
        Debug.Print CDate(s)
    Else
        Debug.Print "无法转换"
    End If
End Sub
```

上述程序返回“无法转换”，因为VBA不认识15点，如果改成15时，则可以转换为日期。

另外，DateSerial和TimeSerial中的参数还可以是负数，例如下面这两行代码。

```
DateSerial(2018,1,-5) 返回日期# 2017/12/26 #
TimeSerial(13,-5,-3) 返回时间#12:54:57#
```

14.5.3 日期时间的格式化

Format函数中的格式字符串中，年、月、日分别用y、m、d表示，时分秒用h、n、s表示。

例如：年份的表达形式中，yyyy表示2018，yy表示18。

其他分量最多是两位数，因此一个字母表示没有前导零。例如mm表示07，而m表示7。

```
Sub 日期时间的格式化输出 ()
    Dim dt As Date
    dt = #7/10/2018 3:02:45 PM#
    Debug.Print Format(dt, "hh:nn:ss yy/mm/dd")
    Debug.Print Format(dt, "aaa")
    Debug.Print Format(dt, "aaaa")
    Debug.Print Format(dt, "ddd")
    Debug.Print Format(dt, "dddd")
End Sub
```

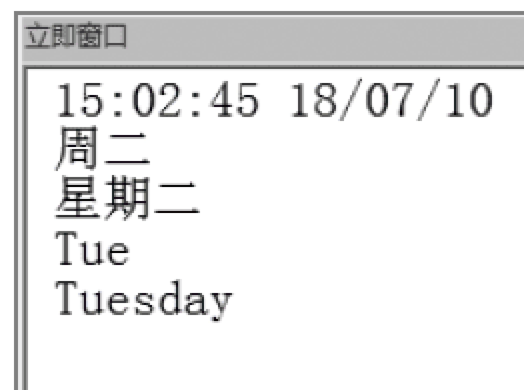


图 14-4　格式化输出日期

运行上述过程，立即窗口的结果如图 14-4 所示。

14.5.4　计算两个日期的差

DateDiff 函数用于计算两个日期的差，必需参数有以下 3 个。

- ❑ Interval：差值的表现形式。
- ❑ Date1：前面的日期。
- ❑ Date2：后面的日期。

下面的程序用来计算两个日期相差的天数。

```
Sub 计算两个日期的差 ()
    Dim dt1 As Date, dt2 As Date
    dt1 = #7/10/2018 1:08:00 PM#
    dt2 = #8/4/2018 5:15:21 AM#
    Debug.Print DateDiff(Interval:="d", Date1:=dt1, Date2:=dt2)
End Sub
```

运行上述程序，返回 25，表示后一个日期比前一个日期大 25 天。

如果把 "d" 换成 "h"，则计算两个日期相差的小时数。

14.5.5　日期与数字的加减

DateAdd 函数用于计算两个日期的差，必需参数有以下 3 个。

- ❑ Interval：差值的表现形式。
- ❑ Number：需要加减的数字，正数负数均可。
- ❑ Date：基准日期。

需要注意的是，DataAdd 函数所需的参数是一个日期和一个数字，结果返回的是另一个日期。

```
Sub 日期与时间的加减 ()
    Dim dt1 As Date, dt2 As Date
    dt1 = #7/10/2018 1:08:00 PM#
    dt2 = DateAdd(Interval:="m", Number:=-10, Date:=dt1)
    Debug.Print dt2
End Sub
```

上述程序的作用是，以 #7/10/2018 1:08:00 PM# 为基准日期，向前推 10 个月，运行结果是：#2017/9/10 13:08:00#。

14.5.6 常见日期信息获取

在实际编程中，经常需要获取基于现在的其他日期。

```
Sub 基于现在的日期()
    Debug.Print "今天", Date
    Debug.Print "昨天", DateAdd("d", -1, Date)
    Debug.Print "明天", DateAdd("d", 1, Date)
    Debug.Print "上周", DateAdd("ww", -1, Date)
    Debug.Print "下周", DateAdd("ww", 1, Date)
    Debug.Print "上月", DateAdd("m", -1, Date)
    Debug.Print "下月", DateAdd("m", 1, Date)
    Debug.Print "去年", DateAdd("yyyy", -1, Date)
    Debug.Print "明年", DateAdd("yyyy", 1, Date)
End Sub
```

运行上述程序，立即窗口的结果如图 14-5 所示。

```
立即窗口
今天          2018/7/7
昨天          2018/7/6
明天          2018/7/8
上周          2018/6/30
下周          2018/7/14
上月          2018/6/7
下月          2018/8/7
去年          2017/7/7
明年          2019/7/7
```

图 14-5 基于今天的其他日期

下面的程序获取基于当前时刻的其他时刻。

```
Sub 基于现在的时间()
    Debug.Print "现在时刻", Now
    Debug.Print "15小时前", DateAdd("h", -15, Now)
    Debug.Print "45分钟后", DateAdd("n", 45, Now)
    Debug.Print "30秒后", DateAdd("s", 30, Now)
End Sub
```

运行上述程序，立即窗口的结果如图 14-6 所示。

```
立即窗口
现在时刻      2018/7/7 8:58:16
15小时前      2018/7/6 17:58:16
45分钟后      2018/7/7 9:43:16
30秒后        2018/7/7 8:58:46
```

图 14-6 今天当前时间的其他时间

此外，还经常遇到计算上月月初、上月月末的日期，以及计算本周一的日期等。

上月月初的计算，需要在现在的基础上减去一个月就可以了，月初始终是 1 号，因此是：

```
DateSerial(Year(Date), Month(Date) - 1, 1)
```

上月月末的计算非常简单，从今天的日期减去今天的天数即可，也就是：

```
Date - Day(Date)
```

也可以写成：

```
DateAdd("d", -Day(Date), Date)
```

本周周一的计算，在今天的基础上减去今天的星期，再加上 1 即可。

```
Date - Weekday(Date, vbMonday) + 1
```

如果要计算本周四的日期，修改为：

```
Date - Weekday(Date, vbMonday) + 4
```

以上程序的源代码文件为“实例文档 100.xlsm”。

14.6　本章小结

随机数在编程开发中具有很重要的地位，例如抽奖、扑克牌游戏的编制都离不开随机数。

在 VBA 中，日期和时间与数字具有对应关系，日期和时间可以与数字直接加减。

两个日期相减的意义是两个日期相隔的天数，两个日期也可以相加、相乘、相除，但没有实际意义。